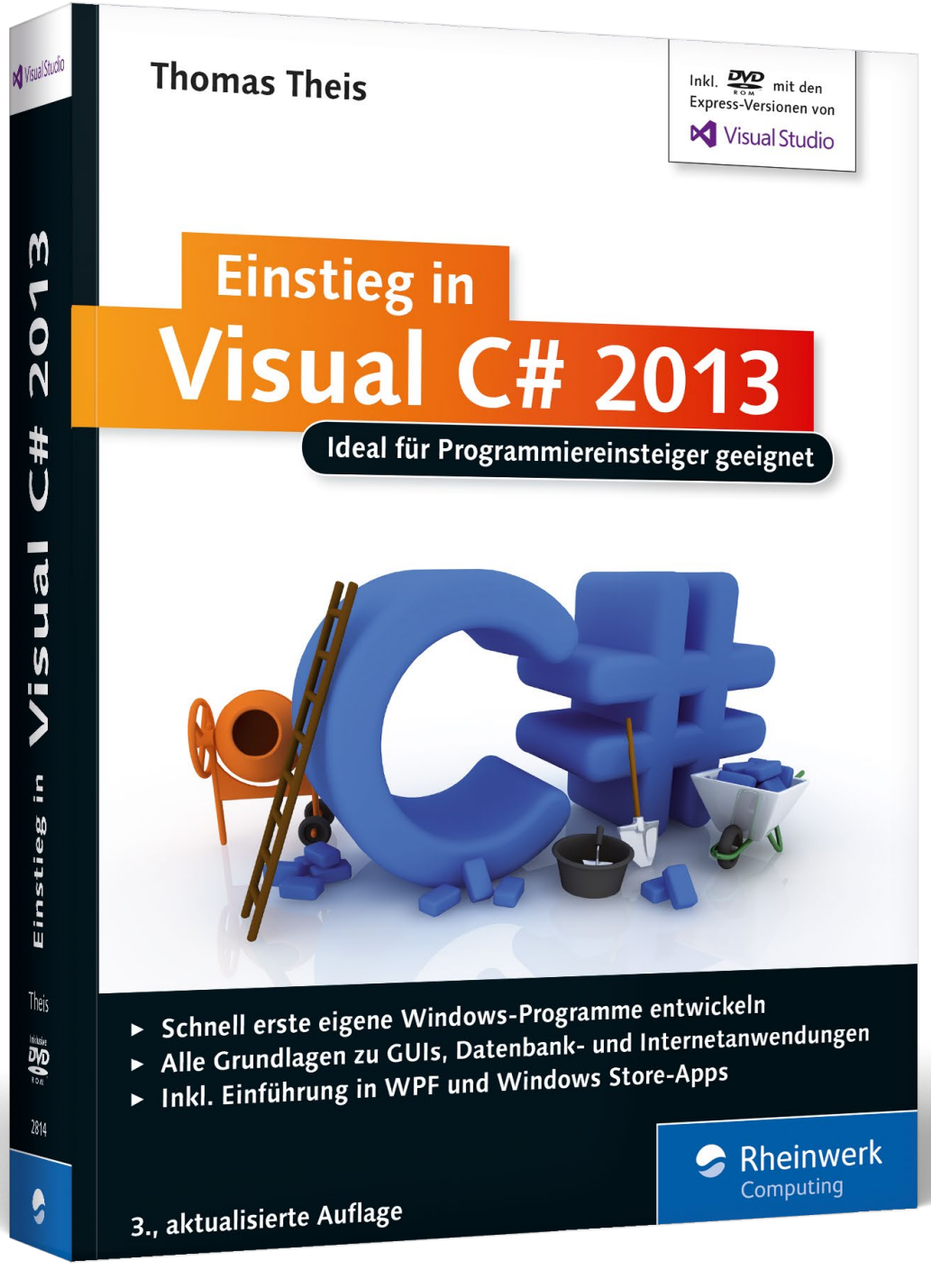


Leseprobe

In diesem Auszug lernen Sie umfassend die Grundlagen der Programmierung kennen: Variablen, Datentypen, Operatoren, Schleifen, Verzweigungen und Steuerelemente. Anschließend programmieren Sie anhand des Beispielprojekts »Tetris» ein Spiel. Zudem enthält diese Leseprobe das vollständige Inhalts- und Stichwortverzeichnis.

Thomas Theis

Einstieg in Visual C# 2013

580 Seiten, broschiert, mit DVD, 3. Auflage 2013
24,90 Euro, ISBN 978-3-8362-2814-5

www.rheinwerk-verlag.de/3573

Kapitel 2
Grundlagen

In diesem Kapitel erlernen Sie auf anschauliche Weise die Sprachgrundlagen von C# in Verbindung mit den gängigen Steuerelementen von Windows-Programmen.

In den folgenden Abschnitten lernen Sie wichtige Elemente der Programmierung, wie Variablen, Operatoren, Verzweigungen und Schleifen, gemeinsam mit wohlbekannten, häufig verwendeten Steuerelementen kennen.

2.1 Variablen und Datentypen

Variablen dienen der vorübergehenden Speicherung von Daten, die sich zur Laufzeit eines Programms ändern können. Eine Variable besitzt einen eindeutigen Namen, unter dem sie angesprochen werden kann.

2.1.1 Namen, Werte

Namensregeln

Für die Namen von Variablen gelten in C# die folgenden Regeln:

- Sie beginnen mit einem Buchstaben.
- Sie können nur aus Buchstaben, Zahlen und einigen wenigen Sonderzeichen (z. B. dem Unterstrich _) bestehen.
- Innerhalb eines Gültigkeitsbereichs darf es keine zwei Variablen mit dem gleichen Namen geben (siehe Abschnitt 2.1.4, »Gültigkeitsbereich«).

Variablen erhalten ihre Werte durch Zuweisung per Gleichheitszeichen. Falls eine Variable als Erstes auf der rechten Seite des Gleichheitszeichens genutzt wird, muss ihr vorher ein Wert zugewiesen werden. Anderenfalls wird ein Fehler gemeldet.

2.1.2 Deklarationen

Neben dem Namen besitzt jede Variable einen Datentyp, der die Art der Information bestimmt, die gespeichert werden kann. Der Entwickler wählt den Datentyp danach aus, ob er Texte, Zahlen ohne Nachkommastellen, Zahlen mit Nachkommastellen oder z. B. logische Werte speichern möchte.

Auswahl des Datentyps

Außerdem muss er sich noch Gedanken über die Größe des Bereichs machen, den die Zahl oder der Text annehmen könnte, und über die gewünschte Genauigkeit bei Zahlen. Im folgenden Abschnitt finden Sie eine Liste der Datentypen.

Variablen müssen in C# immer mit einem Datentyp deklariert werden. Das beugt Fehlern vor, die aufgrund einer falschen Verwendung der Variablen entstehen könnten.

2.1.3 Datentypen

Die folgende Liste enthält die wichtigsten von C# unterstützten Datentypen mit ihrem jeweiligen Wertebereich:

- Datentyp `bool`, Werte `true` oder `false` (*wahr* oder *falsch*)
- Datentyp `byte`, ganze Zahlen von 0 bis 255
- Datentyp `char`, einzelne Zeichen
- Datentyp `decimal`, Gleitkommazahl mit einer Genauigkeit von 28 bis 29 Stellen, Werte von –7,9 mal 10 hoch 28 bis 7,9 mal 10 hoch 28

double

- Datentyp `double`, Gleitkommazahl mit einer Genauigkeit von 15 bis 16 Stellen, Werte von ± 5 mal 10 hoch –324 bis ±1,7 mal 10 hoch 308
- Datentyp `float`, Gleitkommazahl mit einer Genauigkeit von sieben Stellen, Werte von –3,4 mal 10 hoch 38 bis 3,4 mal 10 hoch 38

int

- Datentyp `int`, ganze Zahlen von –2.147.483.648 bis 2.147.483.647
- Datentyp `long`, ganze Zahlen von –9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807
- Datentyp `object`, beliebige Werte
- Datentyp `short`, ganze Zahlen von –32768 bis 32767

string

- Datentyp `string`, Zeichenkette
- benutzerdefinierte Struktur, jedes Element hat seinen eigenen Datentyp und damit seinen eigenen Wertebereich

Im folgenden Beispiel werden Variablen der wichtigsten Typen deklariert, mit Werten versehen und in einem Label angezeigt (Projekt *Datentypen*):

```
private void cmdAnzeige_Click(...)
{
    /* ganze Zahlen */
    byte By;
    short Sh;
    int It;
    long Lg;

    /* Zahlen mit Nachkommastellen */
    double Db1, Db2;
    float Fl;
    decimal De;

    /* Boolesche Variable, Zeichen, Zeichenkette */
    bool Bo;
    char Ch;
    string St;

    By = 200;
    Sh = 30000;
    It = 2000000000;
    Lg = 3000000000;

    Db1 = 1 / 7;
    Db2 = 1.0 / 7;
    Fl = 1.0f / 7;
    De = 1.0m / 7;

    Bo = true;
    Ch = 'a';
    St = "Zeichenkette";

    lblAnzeige.Text =
       "byte: " + By + "\n" +
       "short: " + Sh + "\n" +
       "int: " + It + "\n" +
       "long: " + Lg + "\n" +
```

```
            "double 1: " + Db1 + "\n" +
            "double 2: " + Db2 + "\n" +
            "float: " + Fl + "\n" +
            "decimal: " + De + "\n" +
            "bool: " + Bo + "\n" +
            "char: " + Ch + "\n" +
            "string: " + St;
}
```

Listing 2.1 Projekt »Datentypen«

Das Programm hat nach Betätigung des Buttons die in Abbildung 2.1 dargestellte Ausgabe.

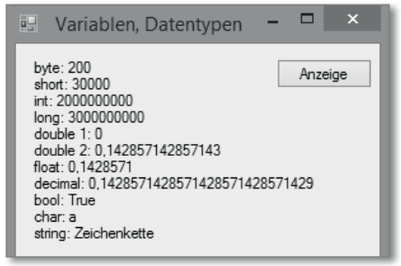


Abbildung 2.1 Wichtige Datentypen

Zur Erläuterung:

- Variablen werden mithilfe von `Datentyp Variable` deklariert.

Wertebereich

- Bei den Zahlen-Datentypen führt eine Über- oder Unterschreitung des Wertebereichs zu einer Fehlermeldung.

Genauigkeit

- Die Datentypen `float`, `double` und `decimal` für Zahlen mit Nachkommastellen unterscheiden sich in ihrer Genauigkeit.
- Bei der Division einer ganzen Zahl durch eine andere ganze Zahl werden die Nachkommastellen abgeschnitten. Zur mathematisch korrekten Division muss einer der beiden Werte als `double`-Zahl gekennzeichnet werden, hier mit 1.0 statt mit 1.
- `float`-Werte müssen mit einem `f` gekennzeichnet werden, `decimal`-Werte mit einem `m`. Damit bekommt die gesamte Division einen `float`- bzw. `decimal`-Wert.

- Werte für den Datentyp `bool` werden mit `true` bzw. `false` zugewiesen, aber mit `True` und `False` ausgegeben.
- Werte für einzelne Zeichen müssen mit einfachen Anführungszeichen, Werte für Zeichenketten mit doppelten Anführungszeichen angegeben werden.

Mehrere Variablen des gleichen Typs können, durch Kommata getrennt, in einer Zeile deklariert werden (z. B. `double Db1, Db2`).

Übung

Übung ÜDatentypen

Schreiben Sie ein Programm, in dem Ihr Nachname, Ihr Vorname, Ihre Adresse, Ihr Alter und Ihr Gehalt jeweils in Variablen eines geeigneten Datentyps gespeichert und anschließend wie in Abbildung 2.2 ausgegeben werden.

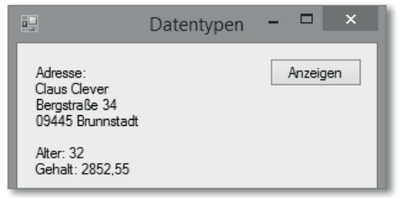


Abbildung 2.2 Übung ÜDatentypen

2.1.4 Gültigkeitsbereich

Lokal

Variablen, die innerhalb einer Methode vereinbart wurden, haben ihre Gültigkeit nur in der Methode. Außerhalb der Methode sind sowohl Name als auch Wert unbekannt. Solche Variablen bezeichnet man auch als lokale Variablen. Sobald die Methode abgearbeitet wurde, steht der Wert auch nicht mehr zur Verfügung. Beim nächsten Aufruf der gleichen Methode werden diese Variablen neu deklariert und erhalten neue Werte.

Klassenweit gültig

Variablen, die außerhalb von Methoden vereinbart werden, sind innerhalb der gesamten Klasse gültig, hier also innerhalb der Klasse des Formulars. Ihr Wert kann in jeder Methode gesetzt oder abgerufen werden und bleibt erhalten, solange das Formular im laufenden Programm existiert.

private

Sie können Variablen auch mit dem Schlüsselwort `private` deklarieren: `private int Mx`. Weitere Einzelheiten zu klassenweit gültigen Variablen finden Sie in Kapitel 5, »Objektorientierte Programmierung«.

public

Variablen, die mit dem Schlüsselwort `public` vereinbart werden, sind *öffentlich*. Damit sind sie auch außerhalb der jeweiligen Klasse, also z. B. in anderen Formularen, gültig. Mehr dazu in Abschnitt 5.2, »Klasse, Eigenschaft, Methode, Objekt«.

Gibt es in einem Programmabschnitt mehrere Variablen mit dem gleichen Namen, gelten folgende Regeln:

- Lokale Variablen mit gleichem Namen in der gleichen Methode sind nicht zulässig.

Ausblenden

- Eine klassenweit gültige Variable wird innerhalb einer Methode von einer lokalen Variablen mit dem gleichen Namen ausgeblendet.

Im folgenden Beispiel werden Variablen unterschiedlicher Gültigkeitsbereiche deklariert, an verschiedenen Stellen verändert und ausgegeben (Projekt *Gültigkeitsbereich*):

```
public partial class Form1 : Form
{
    ...
    int Mx = 0;

    private void cmdAnzeigen1_Click(...)
    {
        int x = 0;
        Mx = Mx + 1;
        x = x + 1;
        lblAnzeige.Text = "x: " + x + "  Mx: " + Mx;
    }

    private void cmdAnzeigen2_Click(...)
    {
        int Mx = 0;
        Mx = Mx + 1;
        lblAnzeige.Text = "Mx: " + Mx;
    }
}
```

Listing 2.2 Projekt »Gültigkeitsbereich«

Zur Erläuterung:

- In der ersten Methode wird der Wert der klassenweit gültigen Variablen `Mx` bei jedem Aufruf erhöht. Die lokale Variable `x` wird immer wieder auf 1 gesetzt (siehe Abbildung 2.3).

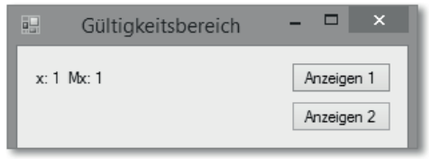


Abbildung 2.3 Lokale und klassenweit gültige Variable

- In der zweiten Methode blendet die lokale Variable `Mx` die gleichnamige klassenweit gültige Variable aus. Die lokale Variable wird immer wieder auf 1 gesetzt (siehe Abbildung 2.4).

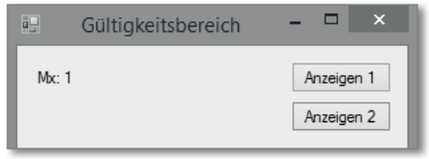


Abbildung 2.4 Lokale Variable

Startwert setzen

Hinweis: Die Variablen wurden vor ihrer ersten Benutzung initialisiert, d. h., sie wurden mit einem Startwert besetzt.

Übung

Übung ÜGültigkeitsbereich

Erstellen Sie ein Programm, in dem zwei Buttons, ein Label und drei Variablen eines geeigneten Datentyps eingesetzt werden:

- die klassenweit gültige Variable `x`
- die Variable `y`, die nur lokal in der Methode zum `Click`-Ereignis des ersten Buttons gültig ist
- die Variable `z`, die nur lokal in der Methode zum `Click`-Ereignis des zweiten Buttons gültig ist

In der ersten Methode werden `x` und `y` jeweils um 0,1 erhöht und angezeigt (siehe Abbildung 2.5).

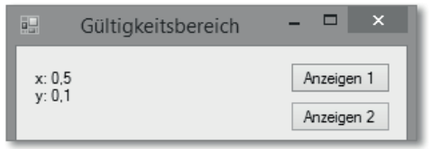


Abbildung 2.5 Ausgabe der ersten Methode nach einigen Klicks

In der zweiten Methode werden `x` und `z` jeweils um 0,1 erhöht und angezeigt (siehe Abbildung 2.6).

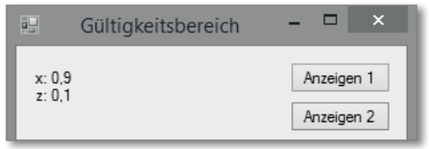


Abbildung 2.6 Ausgabe der zweiten Methode nach weiteren Klicks

2.1.5 Konstanten

Konstanten repräsentieren Werte

Konstanten sind vordefinierte Werte, die während der Laufzeit nicht verändert werden können. Am besten geben Sie Konstanten aussagekräftige Namen, damit sie leichter zu behalten sind als die Werte, die sie repräsentieren. Konstanten werden an einer zentralen Stelle definiert und können an verschiedenen Stellen des Programms genutzt werden. Somit muss eine eventuelle Änderung einer Konstanten zur Entwurfszeit nur an einer Stelle erfolgen. Der Gültigkeitsbereich von Konstanten ist analog zum Gültigkeitsbereich von Variablen.

Integrierte Konstanten

Zu den Konstanten zählen auch die integrierten Konstanten. Auch sie repräsentieren Zahlen, die aber nicht so einprägsam sind wie die Namen der Konstanten.

Im folgenden Beispiel werden mehrere Konstanten vereinbart und genutzt (Projekt *Konstanten*):

```
public partial class Form1 : Form
{
    ...
    const int MaxWert = 75;
    const string Eintrag = "Picture";
```



```
    private void cmdKonstanten_Click(...)
    {
        const int MaxWert = 55;
        const int MinWert = 5;
        lblAnzeige.Text = (MaxWert - MinWert) / 2 +
            "\n" + Eintrag;
    }
}
```

Listing 2.3 Projekt »Konstanten«, Teil 1

Zur Erläuterung:

- Die Konstanten `MaxWert` und `Eintrag` werden mit klassenweiter Gültigkeit festgelegt.
- Innerhalb der Methode werden die beiden lokalen Konstanten `MaxWert` und `MinWert` festgelegt. `MaxWert` blendet die Klassenkonstante gleichen Namens aus, wie Sie in Abbildung 2.7 sehen können.

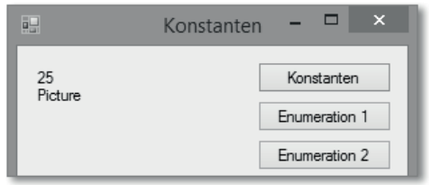


Abbildung 2.7 Konstanten

2.1.6 Enumerationen

Konstanten aufzählen

Enumerationen sind Aufzählungen von Konstanten, die thematisch zusammengehören. Alle Enumerationen haben den gleichen Datentyp, der ganzzahlig sein muss. Bei der Deklaration werden ihnen Werte zugewiesen, am besten explizit. Innerhalb von Visual Studio gibt es für C# zahlreiche vordefinierte Enumerationen. Ähnlich wie bei den integrierten Konstanten sind die Namen der Enumerationen und deren Elemente besser lesbar als die durch sie repräsentierten Zahlen.

Ein Beispiel: Die Enumeration `DialogResult` ermöglicht dem Programmierer, die zahlreichen möglichen Antworten des Benutzers beim Einsatz von

Windows-Standarddialogfeldern (Ja, Nein, Abbrechen, Wiederholen, Ignorieren, ...) anschaulich einzusetzen.

Im folgenden Programm wird mit einer eigenen und einer vordefinierten Enumeration gearbeitet (ebenfalls im Projekt *Konstanten*):

```
public partial class Form1 : Form
{
    ...
    enum Farbe : int
    {
        Rot = 1,
        Gelb = 2,
        Blau = 3
    }

    ...
    private void cmdEnumeration1_Click(...)
    {
        lblAnzeige.Text = "Farbe: " + Farbe.Gelb +
             " " + (int) Farbe.Gelb;
    }

    private void cmdEnumeration2_Click(...)
    {
        lblAnzeige.Text = "Sonntag: " +
            DayOfWeek.Sunday + " " +
            (int) DayOfWeek.Sunday + "\n" +
            "Samstag: " +
            DayOfWeek.Saturday + " " +
            (int) DayOfWeek.Saturday;
    }
}
```

Listing 2.4 Projekt »Konstanten«, Teil 2

Zur Erläuterung:

Klassenweit gültig

- Es wird die Enumeration `Farbe` vom Datentyp `int` vereinbart. Da es sich um einen Typ handelt und nicht um eine Variable oder Konstante, muss sie außerhalb von Methoden vereinbart werden. Damit ist sie automatisch für die gesamte Klasse gültig.

Cast (int)

- In der ersten Ereignismethode wird ein Element der eigenen Enumeration `Farbe` verwendet. Zunächst wird der Name des Elements ausgegeben: `Gelb`. Die Zahl, die das Element repräsentiert, kann erst nach einer Umwandlung in den entsprechenden Datentyp ausgegeben werden. Diese Umwandlung wird mithilfe eines Casts vorgenommen: `(int)` (siehe Abbildung 2.8).

Abbildung 2.8 Erste Enumeration

DayOfWeek

- In der zweiten Ereignismethode werden zwei Elemente der vordefinierten Enumeration `DayOfWeek` verwendet, siehe Abbildung 2.9. Sie können sie zur Ermittlung des Wochentags eines gegebenen Datums verwenden.

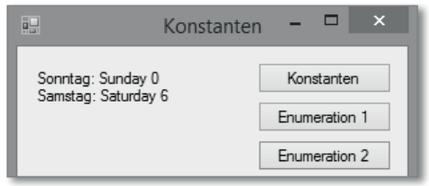


Abbildung 2.9 Zweite Enumeration

2.2 Operatoren

Zum Zusammensetzen von Ausdrücken werden in C#, wie in jeder anderen Programmiersprache auch, Operatoren verwendet. In diesem Buch wurden schon die Operatoren = für Zuweisungen und + für Verkettungen genutzt.

Priorität

Es gibt verschiedene Kategorien von Operatoren. Vorrangregeln (Prioritäten) sind für die Reihenfolge der Abarbeitung zuständig, falls mehrere Operatoren innerhalb eines Ausdrucks verwendet werden. Diese Vorrangregeln finden Sie weiter unten in diesem Abschnitt. Falls Sie sich bei der Verwendung dieser Regeln nicht sicher sind, empfiehlt es sich, durch eigene Klammersetzung die Reihenfolge explizit festzulegen.

2.2.1 Rechenoperatoren

Rechenoperatoren

Rechenoperatoren dienen der Durchführung von Berechnungen, siehe Tabelle 2.1.

Operator	Beschreibung
+	Addition
-	Subtraktion oder Negation
*	Multiplikation
/	Division
%	Modulo
++	Erhöhung um 1
--	Verminderung um 1

Tabelle 2.1 Rechenoperatoren

Ganzzahldivision

Bei der Division von zwei ganzen Zahlen sollten Sie beachten, dass die Nachkommastellen abgeschnitten werden. Möchten Sie das nicht, müssen Sie zumindest eine der beiden Zahlen als Zahl mit Nachkommastellen kennzeichnen, z. B. durch Anhängen von `.0`: Statt `5` schreiben Sie `5.0`.

Modulo

Der Modulo-Operator `%` berechnet den Rest einer Division. Einige Beispiele sehen Sie in Tabelle 2.2.

Ausdruck	Ergebnis	Erklärung
`19 % 4`	3	19 durch 4 ist 4 Rest 3
`19.5 % 4.2`	2.7	19,5 durch 4,2 ist 4 Rest 2,7

Tabelle 2.2 Modulo-Operator

++, --

Die Operatoren ++ und -- dienen als Schreibabkürzung und sollen mithilfe des Projekts *Rechenoperatoren* erläutert werden:

```
private void cmdAnzeigen1_Click(...)
{
    int x = 5;
    x++;
    ++x;
    x = x + 1;
    lblA.Text = "Ergebnis: " + x;
}

private void cmdAnzeigen2_Click(...)
{
    int x = 5;
    lblA.Text = "Ergebnis: " + x++;
}

private void cmdAnzeigen3_Click(...)
{
    int x = 5;
    lblA.Text = "Ergebnis: " + ++x;
}
```

Listing 2.5 Projekt »Rechenoperatoren«

Zur Erläuterung:

- In der ersten Methode hat `x` zunächst den Wert 5. Der Wert kann mit `++x` oder mit `x++` oder mit `x = x + 1` jeweils um 1 erhöht werden. Anschließend hat `x` den Wert 8.
- In der zweiten Methode wird `x` zunächst ausgegeben und anschließend um 1 erhöht. Das liegt daran, dass der Operator ++ hinter `x` steht. In der Ausgabe sehen Sie den alten Wert 5, `x` hat nach der Anweisungszeile den Wert 6. **x++**
- In der dritten Methode wird `x` zunächst um 1 erhöht und anschließend ausgegeben. Das liegt daran, dass der Operator ++ vor `x` steht. In der Ausgabe sehen Sie den neuen Wert 6, `x` hat nach der Anweisungszeile ebenfalls den Wert 6. **++x**
- Die Schreibweise `x = x + 1;` als eigene Anweisungszeile schafft hier Klarheit. **x=x+1**
- Für den Operator `--` gilt sinngemäß das Gleiche.

Von links nach rechts

Multiplikation und Division innerhalb eines Ausdrucks sind gleichrangig und werden von links nach rechts in der Reihenfolge ihres Auftretens ausgewertet. Dasselbe gilt für Additionen und Subtraktionen, die zusammen

in einem Ausdruck auftreten. Multiplikation und Division werden vor Addition und Subtraktion ausgeführt.

Klammern

Mit Klammern kann diese Rangfolge außer Kraft gesetzt werden, damit bestimmte Teilausdrücke vor anderen Teilausdrücken ausgewertet werden. In Klammern gesetzte Operationen haben grundsätzlich Vorrang. Innerhalb der Klammern gilt jedoch wieder die normale Rangfolge der Operatoren.

Projekt

Im Projekt *Rechenoperatoren* können Sie auch die beiden Berechnungen mit dem Operator % nachvollziehen.

Übung

Übung ÜRechenoperatoren

Berechnen Sie die beiden folgenden Ausdrücke, speichern Sie das Ergebnis in einer Variablen eines geeigneten Datentyps, und zeigen Sie es an:

- 1. Ausdruck: `3 * -2.5 + 4 * 2`
- 2. Ausdruck: `3 * (-2.5 + 4) * 2`

2.2.2 Vergleichsoperatoren

Vergleich

Vergleichsoperatoren (siehe Tabelle 2.3) dienen dazu, festzustellen, ob bestimmte Bedingungen zutreffen oder nicht. Das Ergebnis nutzt man u. a. zur Ablaufsteuerung von Programmen. In Abschnitt 2.4, »Verzweigungen«, wird hierauf genauer eingegangen.

Operator	Beschreibung
`<`	kleiner als
`<=`	kleiner als oder gleich
`>`	größer als
`>=`	größer als oder gleich
`==`	gleich
`!=`	ungleich

Tabelle 2.3 Vergleichsoperatoren

Einige Beispiele sehen Sie in Tabelle 2.4.

Ausdruck	Ergebnis
`5 > 3`	`true`
`3 == 3.2`	`false`
`5 + 3 * 2 >= 12`	`false`
`"Maier" == "Mayer"`	`false`

Tabelle 2.4 Nutzung von Vergleichsoperatoren

Projekt

Alle Vergleiche innerhalb dieses Abschnitts können Sie auch mithilfe des Codes im Projekt *Vergleichsoperatoren* nachvollziehen.

Übung

Übung ÜVergleichsoperatoren

Ermitteln Sie das Ergebnis der beiden folgenden Ausdrücke, speichern Sie es in einer Variablen eines geeigneten Datentyps, und zeigen Sie es an:

- 1. Ausdruck: `12 – 3 >= 4 * 2.5`
- 2. Ausdruck: `"Maier" != "Mayer"`

2.2.3 Logische Operatoren

Logik

Logische Operatoren dienen dazu, mehrere Bedingungen zusammenzufassen. Das Ergebnis nutzt man ebenfalls u. a. zur Ablaufsteuerung von Programmen (siehe hierzu auch Abschnitt 2.2.4, »Verkettungsoperator«). Die logischen Operatoren sehen Sie in Tabelle 2.5.

Operator	Beschreibung	Das Ergebnis ist true, wenn …
`!`	Nicht	… der Ausdruck `false` ist.
`&&`	Und	… beide Ausdrücke `true` sind.
`||`	inklusives Oder	… mindestens ein Ausdruck `true` ist.
`^`	exklusives Oder	… genau ein Ausdruck `true` ist.

Tabelle 2.5 Logische Operatoren

! && || ^

Es seien die Variablen `A = 1`, `B = 3` und `C = 5` gesetzt. Die Ausdrücke in der ersten Spalte von Tabelle 2.6 ergeben dann jeweils die Ergebnisse in der zweiten Spalte.

Ausdruck	Ergebnis
`!(A < B)`	`false`
`(B > A) && (C > B)`	`true`
`(B < A) || (C < B)`	`false`
`(B < A) ^ (C > B)`	`true`

Tabelle 2.6 Ausdrücke mit logischen Operatoren

Projekt

Alle Berechnungen innerhalb dieses Abschnitts können Sie auch mithilfe des Codes im Projekt *LogischeOperatoren* nachvollziehen.

Übung

Übung ÜLogische-Operatoren

Ermitteln Sie das Ergebnis der beiden folgenden Ausdrücke, speichern Sie es in einer Variablen eines geeigneten Datentyps, und zeigen Sie es an:

- 1. Ausdruck: `4 > 3 && –4 > –3`
- 2. Ausdruck: `4 > 3 || –4 > –3`

&, |

Sie können auch die logischen Operatoren `&` (statt `&&`) und `|` (statt `||`) verwenden. Hierbei werden alle Teile des Vergleichsausdrucks ausgewertet. Im Gegensatz dazu wird bei den Operatoren `&&` und `||` die Auswertung abgebrochen, sobald sich der Wert des Ausdrucks nicht mehr verändern kann. Die Ergebnisse unterscheiden sich allerdings nur dann, wenn innerhalb des Vergleichsausdrucks Werte verändert werden, z. B. mit den Operatoren `++` oder `--`.

2.2.4 Verkettungsoperator

Umwandlung in String

Der Operator `+` dient der Verkettung von Zeichenfolgen. Ist einer der Ausdrücke keine Zeichenfolge, sondern eine Zahl, wird er (wenn möglich) in eine Zeichenfolge verwandelt. Das Gesamtergebnis ist dann wiederum eine Zeichenfolge. Beispiel:


```
private void cmdAnzeige_Click(...)
{
    string a, b;
    double d;
    int x;
    b = "Hallo";
    d = 4.6;
    x = -5;
    a = b + " Welt " + d + " " + x + " " + 12;
    lblAnzeige.Text = a;
    // lblAnzeige.Text = x;
}
```

Listing 2.6 Projekt »Verkettungsoperator«

Zur Erläuterung:

- Die Zeichenkette `a` wird aus Variablen und Werten unterschiedlichen Datentyps zusammengesetzt.
- Die letzte Anweisung wurde auskommentiert, weil sie zu einem Fehler führt. Die `int`-Variable `x` kann nicht direkt als Wert für die Eigenschaft `Text` verwendet werden. Sie muss zunächst umgewandelt werden.
- Das Ergebnis ist in Abbildung 2.10 zu sehen.
- Ein weiteres Beispiel stand bereits in Abschnitt 1.4.5, »Das Codefenster«.

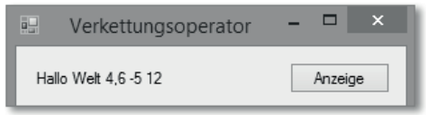


Abbildung 2.10 Verkettung

2.2.5 Zuweisungsoperatoren

Zeichen =

Den einfachsten Zuweisungsoperator, das Gleichheitszeichen, haben Sie bereits genutzt. Es gibt zur Verkürzung von Anweisungen noch weitere Zuweisungsoperatoren. Eine Auswahl sehen Sie in Tabelle 2.7.

Operator	Beispiel	Ergebnis
=	x = 7	x erhält den Wert 7.
+=	x += 5	Der Wert von x wird um 5 erhöht.
-=	x -= 5	Der Wert von x wird um 5 verringert.
*=	x *= 3	Der Wert von x wird auf das Dreifache erhöht.
/=	x /= 3	Der Wert von x wird auf ein Drittel verringert.
%=	x %= 3	x wird durch 3 geteilt, der Rest der Division wird x zugewiesen.
+=	z += "abc"	Die Zeichenkette z wird um den Text abc verlängert.

Tabelle 2.7 Zuweisungsoperatoren

2.2.6 Rangfolge der Operatoren

Priorität

Enthält ein Ausdruck mehrere Operationen, werden die einzelnen Teilausdrücke in einer bestimmten Rangfolge ausgewertet und aufgelöst, die als Rangfolge bzw. Priorität der Operatoren bezeichnet wird. Es gilt die Rangfolge in Tabelle 2.8.

Operator	Beschreibung
- !	negatives Vorzeichen, logisches Nicht
* / %	Multiplikation, Division, Modulo
+ -	Addition, Subtraktion
< > <= >=	Vergleichsoperatoren für kleiner und größer
== !=	Vergleichsoperatoren für gleich und ungleich
&&	logisches Und
||	logisches Oder

Tabelle 2.8 Rangfolge der Operatoren

Je weiter oben die Operatoren in der Tabelle stehen, desto höher ist ihre Priorität.

Klammern

Wie schon bei den Rechenoperatoren erwähnt: Mit Klammern kann diese Rangfolge außer Kraft gesetzt werden, damit bestimmte Teilausdrücke vor anderen Teilausdrücken ausgewertet werden. In Klammern gesetzte Operationen haben grundsätzlich Vorrang. Innerhalb der Klammern gilt jedoch wieder die normale Rangfolge der Operatoren.

Übung

Übung ÜOperatoren

Sind die Bedingungen in Tabelle 2.9 wahr oder falsch? Lösen Sie die Aufgabe möglichst ohne PC.

Nr.	Werte	Bedingung
1	a=5 b=10	a>0 && b!=10
2	a=5 b=10	a>0 || b!=10
3	z=10 w=100	z!=0 || z>w || w-z==90
4	z=10 w=100	z==11 && z>w || w-z==90
5	x=1.0 y=5.7	x>=.9 && y<=5.8
6	x=1.0 y=5.7	x>=.9 && !(y<=5.8)
7	n1=1 n2=17	n1>0 && n2>0 || n1>n2 && n2!=17
8	n1=1 n2=17	n1>0 && (n2>0 || n1>n2) && n2!=17

Tabelle 2.9 Übung ÜOperatoren

2.3 Einfache Steuerelemente

Windows-Programmierung mit Visual C# besteht aus zwei Teilen: der Arbeit mit visuellen Steuerelementen und der Programmierung mit der Sprache. Beides soll in diesem Buch parallel vermittelt werden, damit die eher theoretischen Abschnitte zur Programmiersprache durch eine anschauliche Praxis vertieft werden können.

Daher wird in diesem Abschnitt mit vier weiteren Steuerelementen gearbeitet, bevor im nächsten Abschnitt die Verzweigungen zur Programmsteuerung vorgestellt werden: den Steuerelementen Panel, Zeitgeber, Textfeld und Zahlenauswahlfeld.

2.3.1 Panel

Container

Ein Panel dient normalerweise als Container für andere Steuerelemente. In diesem Abschnitt wird es zur visuellen Darstellung eines Rechtecks und für eine kleine Animation genutzt.

Die Eigenschaften `BackColor` (Hintergrundfarbe), `Location` (Position) und `Size` (Größe) sind Ihnen schon von anderen Steuerelementen bekannt.

Mithilfe des nachfolgenden Programms im Projekt *Panel* wird ein Panel durch Betätigung von vier Buttons um 10 Pixel nach oben, unten, links oder rechts verschoben. Es hat die Größe 100 × 100 Pixel, die Startposition `X=145` und `Y=80` sowie eine eigene Hintergrundfarbe. Die Bewegung wird mithilfe der Struktur `Point` durchgeführt.

In Abbildung 2.11 und Abbildung 2.12 ist das Panel im Startzustand bzw. nach einigen Klicks zu sehen.

Abbildung 2.11 Panel, Startzustand

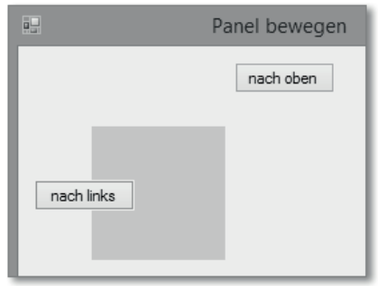


Abbildung 2.12 Panel, nach ein paar Klicks

Der Programmcode:

```
private void cmdOben_Click(...)
{
    p.Location = new Point(
        p.Location.X, p.Location.Y - 10);
}

private void cmdLinks_Click(...)
{
    p.Location = new Point(
        p.Location.X - 10, p.Location.Y);
}

private void cmdRechts_Click(...)
{
    p.Location = new Point(
        p.Location.X + 10, p.Location.Y);
}

private void cmdUnten_Click(...)
{
    p.Location = new Point(
        p.Location.X, p.Location.Y + 10);
}
```

Listing 2.7 Projekt »Panel«

2.3.2 Zeitgeber

Timer-Intervall
Enabled

Ein Zeitgeber (Timer) erzeugt in festgelegten Abständen Zeittakte. Diese Zeittakte sind Ereignisse, die der Entwickler mit Aktionen verbinden kann. Das zugehörige Ereignis heißt `Tick`. Ein Zeitgeber kann wie jedes andere Steuerelement zum Formular hinzugefügt werden. Da es sich aber um ein nicht sichtbares Steuerelement handelt, wird er unterhalb des Formulars angezeigt. Auch zur Laufzeit ist er nicht sichtbar. Seine wichtigste Eigenschaft ist das Zeitintervall, in dem das Ereignis auftreten soll. Dieses Zeitintervall wird in Millisekunden angegeben. Die Eigenschaft `Enabled` dient der Aktivierung bzw. Deaktivierung des Zeitgebers. Sie können sie zur Entwicklungszeit oder zur Laufzeit auf `true` oder `false` stellen.

Im nachfolgenden Programm im Projekt *Zeitgeber* erscheint zunächst ein Formular mit zwei Buttons. Betätigen Sie den START-Button, erscheint ein *x* in einem Bezeichnungsfeld. Alle 0,5 Sekunden erscheint automatisch ein weiteres *x*, siehe Abbildung 2.13. Das wird durch den Timer gesteuert, bei dem der Wert für die Eigenschaft `Interval` auf 500 gesetzt wurde. Nach Betätigung des STOP-Buttons kommt kein weiteres *x* hinzu.

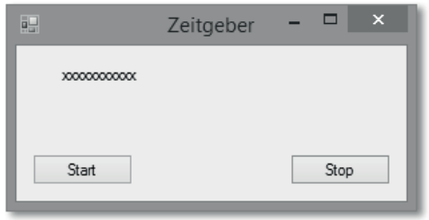


Abbildung 2.13 Nach einigen Sekunden

Der zugehörige Code:

```
private void cmdStart_Click(...)
{
    timAnzeige.Enabled = true;
}

private void cmdStop_Click(...)
{
    timAnzeige.Enabled = false;
}
```



```
private void timAnzeige_Tick(...)
{
    lblAnzeige.Text += "x";
}
```

Listing 2.8 Projekt »Zeitgeber«

Übung

Übung ÜPanel-Zeitgeber

Erstellen Sie eine Windows-Anwendung. In der Mitte eines Formulars sollen zu Beginn vier Panels verschiedener Farbe der Größe 20 × 20 Pixel platziert werden, siehe Abbildung 2.14.

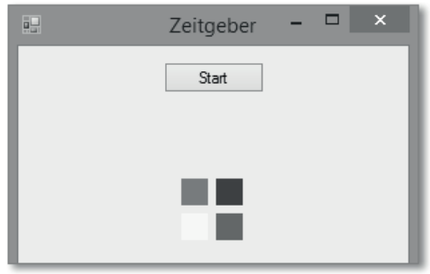


Abbildung 2.14 Startzustand

Sobald ein START-Button betätigt wird, sollen sich diese vier Panels diagonal in ca. fünf bis zehn Sekunden zu den Ecken des Formulars bewegen, jedes Panel in eine andere Ecke (siehe Abbildung 2.15).

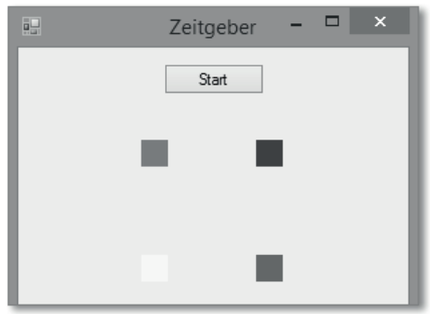


Abbildung 2.15 Nach einigen Sekunden

Übung

Übung ÜKran

Diese Übung gehört nicht zum Pflichtprogramm. Sie ist etwas umfangreicher, verdeutlicht aber die Möglichkeiten einer schnellen Visualisierung von Prozessen durch Visual C# mit wenigen Programmzeilen.

Konstruieren Sie aus mehreren Panels einen Kran (Fundament, senkrechtes Hauptelement, waagerechter Ausleger, senkrechter Haken am Ausleger). Der Benutzer soll die Möglichkeit haben, über insgesamt acht Buttons die folgenden Aktionen auszulösen:

- Haken um 10 Pixel ausfahren bzw. einfahren
- Ausleger um 10 Pixel ausfahren bzw. einfahren
- Kran um 10 Pixel nach rechts bzw. links fahren
- Kran um 10 Pixel in der Höhe ausfahren bzw. einfahren

Denken Sie daran, dass bei vielen Bewegungen mehrere Steuerelemente bewegt werden müssen, da der Kran sonst seinen Zusammenhalt verliert. Manche Aktionen resultieren nur aus Größenveränderungen (Eigenschaften `Width` und `Height`), andere nur aus Ortsveränderungen (`Location`), wieder andere aus beidem. In Abbildung 2.16 und Abbildung 2.17 sehen Sie den Kran im Startzustand bzw. nach einigen Klicks.

Es können natürlich immer noch widersprüchliche Bewegungen auftreten. Mit zunehmendem Programmierwissen können Sie diesen Problemen später noch abhelfen.

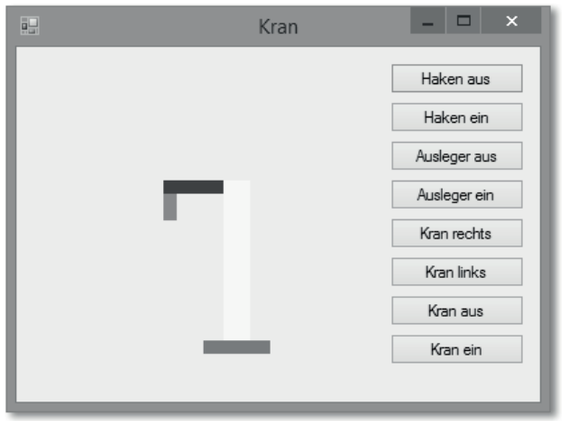


Abbildung 2.16 Startzustand

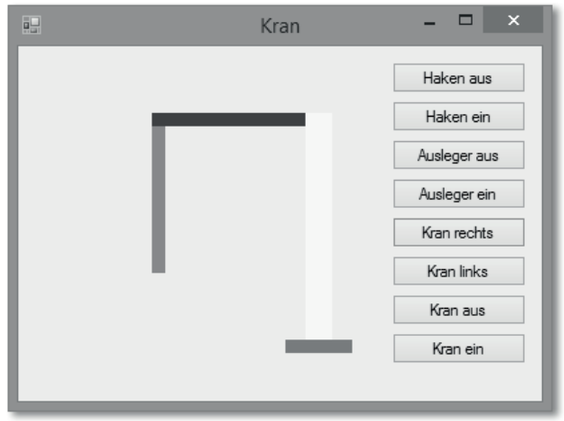


Abbildung 2.17 Nach einigen Aktionen

2.3.3 Textfelder

Eingabefeld

Ein Textfeld dient in erster Linie dazu, die Eingabe von Text oder Zahlen vom Benutzer entgegenzunehmen. Diese Eingaben werden in der Eigenschaft `Text` des Textfelds gespeichert. Das Aussehen und das Verhalten eines Textfelds werden u. a. durch folgende Eigenschaften gekennzeichnet:

- `MultiLine`: Steht `MultiLine` auf `true`, können Sie bei der Eingabe und bei der Anzeige mit mehreren Textzeilen arbeiten.
- `ScrollBars`: Sie können ein Textfeld mit vertikalen und/oder horizontalen Bildlaufleisten zur Eingabe und Anzeige längerer Texte versehen.
- `MaxLength`: Mit dieser Eigenschaft können Sie die Anzahl der Zeichen des Textfelds beschränken. Ist keine Beschränkung vorgesehen, kann das Textfeld 32768 Zeichen aufnehmen.
- `PasswordChar`: Falls Sie für diese Eigenschaft im Entwurfsmodus ein Platzhalterzeichen eingegeben haben, wird während der Laufzeit für jedes eingegebene Zeichen nur dieser Platzhalter angezeigt. Diese Eigenschaft wird vor allem bei Passwortabfragen verwendet.

Passwort

Der Inhalt eines Textfelds kann mit den gewohnten Mitteln (z. B. Strg + C und Strg + V) in die Zwischenablage kopiert bzw. aus der Zwischenablage eingefügt werden.

Im nachfolgenden Programm im Projekt *Textfelder* kann der Benutzer in einem Textfeld einen Text eingeben. Nach Betätigung des Buttons AUSGABE wird der eingegebene Text in einem zusammenhängenden Satz ausgegeben (siehe Abbildung 2.18).

Abbildung 2.18 Eingabe in Textfeld

Der Code lautet wie folgt:

```
private void cmdAusgabe_Click(...)
{
    lblAusgabe.Text = "Sie haben '" +
        txtEingabe.Text + "' eingegeben";
}
```

Listing 2.9 Projekt »Textfelder«

Zur Erläuterung:

- In der Eigenschaft `Text` des Textfelds wird die Eingabe gespeichert. Die Eigenschaft wird in einen längeren Ausgabetext eingebettet.

Zahlen eingeben

Bei der Eingabe und Auswertung von Zahlen sind einige Besonderheiten zu beachten. Im nachfolgenden Programm, ebenfalls im Projekt *Textfelder*, kann der Benutzer in einem Textfeld eine Zahl eingeben. Nach Betätigung des Buttons RECHNEN wird der Wert dieser Zahl verdoppelt, das Ergebnis wird in einem Label darunter ausgegeben:

```
private void cmdRechnen_Click(...)
{
    double wert;
    wert = Convert.ToDouble(txtEingabe.Text);
    wert = wert * 2;
    lblAusgabe.Text = "Ergebnis: " +  wert;
}
```

Listing 2.10 Projekt »Textfelder«, Zahleneingabe

Zur Erläuterung:

ToDouble()

Es muss dafür gesorgt werden, dass der Inhalt des Textfelds explizit in eine Zahl (mit möglichen Nachkommastellen) umgewandelt wird. Das erreichen Sie mithilfe der Methode `ToDouble()` aus der Klasse `Convert`. Die Klasse `Convert` bietet eine Reihe von Methoden für die Umwandlung (= Konvertierung) in andere Datentypen.

- Wenn eine Zeichenkette eingegeben wurde, die eine Zahl darstellt, wird sie auf die oben angegebene Weise in eine Zahl umgewandelt, mit der dann gerechnet werden kann.
- Stellt die eingegebene Zeichenkette keine Zahl dar, kommt es zu einem Laufzeitfehler. Diese Situation sollten Sie natürlich vermeiden:
  - Sie können vorher überprüfen, ob es sich bei der Zeichenkette um eine gültige Zahl handelt, und entsprechend reagieren. Das wird Ihnen möglich sein, sobald Sie Verzweigungen zur Programmsteuerung beherrschen.
  - **Ausnahmebehandlung** Allgemein können Sie Programme so schreiben, dass ein Programmabbruch abgefangen werden kann. Das wird Ihnen möglich sein, sobald Sie die Ausnahmebehandlung (siehe hierzu Kapitel 3, »Fehlerbehandlung«) beherrschen.

Einige Beispiele:

Abbildung 2.19 zeigt die Eingabe einer Zahl mit Nachkommastellen.

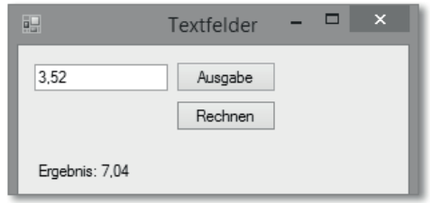


Abbildung 2.19 Eingabe einer Zahl mit Nachkommastellen

Debugging beenden

Die Eingabe einer Zeichenkette, z. B. »abc«, führt zur Anzeige einer nicht behandelten Ausnahme. Die Zeile, in der der Fehler auftritt, wird im Code markiert, damit der Fehler beseitigt werden kann (siehe Abbildung 2.20). Es wird ein zusätzliches Dialogfeld angezeigt. Wenn man darin den Button WEITER betätigt, wird das Programm beendet. Wählt man den Button UNTERBRECHEN, wird das Programm unterbrochen. Bei einem unterbro-

chenen Programm können Sie die aktuellen Werte von Variablen kontrollieren, indem Sie die Maus über dieser Variablen platzieren. Anschließend muss das Programm über den Menüpunkt DEBUGGEN • DEBUGGING BEENDEN beendet werden, bevor es neu gestartet werden kann.

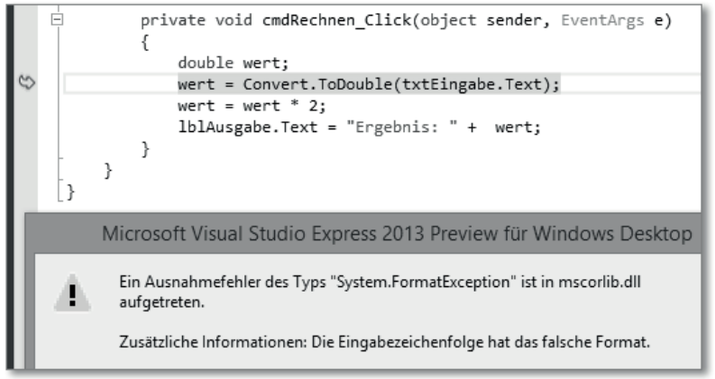


Abbildung 2.20 Markierung der Fehlerzeile

Die Eingabe einer Zahl, bei der ein Punkt statt eines Kommas zur Abtrennung von Nachkommastellen eingegeben wird, führt zu einem ganz anderen Rechenergebnis, siehe Abbildung 2.21. Der Punkt wird ignoriert, die Zahl wird als 352 angesehen und führt so zu dem Ergebnis 704.

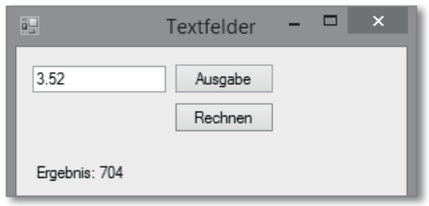


Abbildung 2.21 Punkt vor den Nachkommastellen

2.3.4 Zahlenauswahlfeld

NumericUpDown

Das Steuerelement *Zahlenauswahlfeld* (`NumericUpDown`) bietet eine andere Möglichkeit, Zahlenwerte an ein Programm zu übermitteln. Die Zahlenwerte können innerhalb selbst gewählter Grenzen und in selbst definierten Schritten über zwei kleine Pfeiltasten ausgewählt werden. Sie können aber auch weiterhin wie bei einem Textfeld eingegeben werden.

Wichtige Eigenschaften des Steuerelements sind:

Value

- `Value`: bezeichnet zur Entwicklungszeit den Startwert und zur Laufzeit den vom Benutzer aktuell eingestellten Wert.
- `Maximum`, `Minimum`: bestimmen den größtmöglichen Wert und den kleinstmöglichen Wert der Eigenschaft `Value`. Es handelt sich also um die Werte, die durch die Auswahl mit den Pfeiltasten ganz oben und ganz unten erreicht werden können.
- `Increment`: Mit `Increment` wird die Schrittweite eingestellt, mit der sich der Wert (Eigenschaft `Value`) ändert, wenn der Benutzer eine der kleinen Pfeiltasten betätigt.
- `DecimalPlaces`: bestimmt die Anzahl der Nachkommastellen in der Anzeige des Zahlenauswahlfelds.

ValueChanged

Das wichtigste Ereignis dieses Steuerelements ist `ValueChanged`. Es tritt bei der Veränderung der Eigenschaft `Value` ein und sollte anschließend zur Programmsteuerung verwendet werden.

Im nachfolgenden Programm im Projekt *Zahlenauswahlfeld* werden alle diese Eigenschaften und das genannte Ereignis genutzt. Der Benutzer kann Zahlenwerte zwischen –5,0 und +5,0 in Schritten von 0,1 über ein Zahlenauswahlfeld einstellen. Der ausgewählte Wert wird unmittelbar in einem Label angezeigt (siehe Abbildung 2.22).

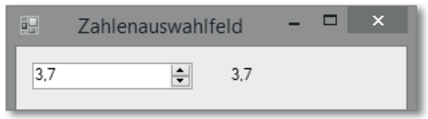


Abbildung 2.22 Zahlenauswahlfeld

Die Eigenschaften wurden zur Entwicklungszeit wie folgt eingestellt:

- `Value`: Wert 2, die Anwendung startet also bei dem Wert 2,0 für das Zahlenauswahlfeld
- `Maximum`, `Minimum`: Werte –5 und +5
- `Increment`: Wert 0,1
- `DecimalPlaces`: Wert 1 zur Anzeige einer einzelnen Nachkommastelle

Der Code lautet:

```
private void numEingabe_ValueChanged(...)
{
    lblAusgabe.Text = "Wert: " + numEingabe.Value;
}
```

Listing 2.11 Projekt »Zahlenauswahlfeld«

2.4 Verzweigungen

Der Programmcode wurde bisher rein sequenziell abgearbeitet, d. h. eine Anweisung nach der anderen. Kontrollstrukturen ermöglichen eine Steuerung dieser Reihenfolge. Die Kontrollstrukturen unterteilen sich in Verzweigungen und Schleifen. Verzweigungen gestatten dem Programm, in verschiedene alternative Anweisungsblöcke zu verzweigen.

Es gibt die beiden Verzweigungsstrukturen `if...else` und `switch...case`. Diese Auswahlmöglichkeiten übergeben aufgrund von Bedingungen die Programmausführung an einen bestimmten Anweisungsblock. Bedingungen werden mithilfe der bereits vorgestellten Vergleichsoperatoren erstellt.

2.4.1 if...else

Eine Verzweigung mit `if...else` hat folgenden Aufbau:

```
if (Bedingung)
{
   Anweisungen1
}
[ else
{
    Anweisungen2
} ]
```

if...else

Die Bedingung wird ausgewertet, sie ist entweder wahr oder falsch (`true` oder `false`). Ist die Bedingung wahr, wird der erste Teil (`Anweisungen1`) ausgeführt. Ist die Bedingung nicht wahr und gibt es einen `else`-Teil, wird dieser Teil (`Anweisungen2`) ausgeführt. Der `else`-Teil ist optional.

Ohne Klammern

Falls es sich bei `AnweisungenX` nur um eine einzelne Anweisung handelt, können in diesem Teil der Verzweigung die geschweiften Klammern weggelassen werden.

Geschachtelt

Verzweigungen können auch ineinander verschachtelt werden, falls es mehr als zwei Möglichkeiten für den weiteren Programmverlauf gibt.

Mehrere Vergleiche

Eine Bedingung kann aus einem einfachen Ausdruck mit Vergleichsoperatoren bestehen oder aus mehreren Vergleichsausdrücken.

Es folgen einige Beispiele im Projekt *IfElse*, siehe Abbildung 2.23. Die untersuchten Zahlenwerte können über Zahlenauswahlfelder eingestellt werden. Testen Sie die Möglichkeiten durch unterschiedliche Einstellungen der Zahlenauswahlfelder, bevor Sie einen der Buttons betätigen.

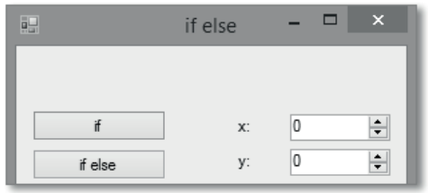


Abbildung 2.23 Projekt »IfElse«

Zunächst ein `if` ohne `else`:

```
private void cmdAnzeige1_Click(...)
{
    int x = (int) numX.Value;
    lblAnzeige.Text = "";

    if (x > 0)
    {
        lblAnzeige.Text = "x ist größer als 0";
        numX.BackColor = Color.LightGreen;
    }
}
```

Listing 2.12 Projekt »IfElse«, Teil 1

Zur Erläuterung:

- Die `int`-Variable `x` erhält den Wert, der im Zahlenauswahlfeld `numX` eingestellt wurde. Da dieses Feld eine Variable vom Typ `decimal` liefert, muss der Wert zunächst mithilfe des Casts `(int)` umgewandelt werden.
- Das Label wird geleert.
- Nun zur eigentlichen Verzweigung: Falls der Wert von `x` größer als 0 ist, wird ein entsprechender Text ausgegeben. Außerdem wird das Zahlenauswahlfeld hellgrün eingefärbt (siehe Abbildung 2.24).
- Da es sich um zwei Anweisungen handelt, müssen sie in geschweifte Klammern gesetzt werden.
- Falls der Wert von `x` kleiner oder gleich 0 ist, passiert nichts. Das Label bleibt leer. Es gibt keinen `else`-Teil, in dem etwas ausgeführt werden könnte.

Einrückung

- Die Anweisungen innerhalb des `if`-Blocks werden eingerückt. Das Programm ist dadurch leichter lesbar. Das ist eine empfehlenswerte Vorgehensweise, insbesondere bei weiteren Verschachtelungen innerhalb des Programms.

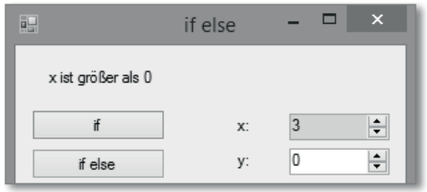


Abbildung 2.24 Bedingung trifft zu

Nun folgt ein `if` mit `else`. Es wird also in jedem Fall etwas ausgeführt:

```
private void cmdAnzeige2_Click(...)
{
    int x = (int) numX.Value;

    if (x > 0)
    {
        lblAnzeige.Text = "x ist größer als 0";
        numX.BackColor = Color.LightGreen;
    }
    else
```



```
    {
        lblAnzeige.Text =
            "x ist kleiner als 0 oder gleich 0";
        numX.BackColor = Color.LightBlue;
    }
}
```

Listing 2.13 Projekt »IfElse«, Teil 2

Zur Erläuterung:

- Falls der Wert von `x` jetzt kleiner oder gleich 0 ist, wird auch etwas ausgegeben. Außerdem wird das Zahlenauswahlfeld nunmehr hellblau eingefärbt (siehe Abbildung 2.25).
- Da es sich auch im `else`-Teil um zwei Anweisungen handelt, müssen sie ebenfalls in geschweifte Klammern gesetzt werden.

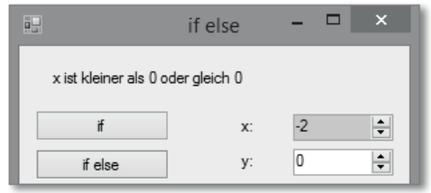


Abbildung 2.25 Zweig mit else

Es folgt ein Beispiel mit drei möglichen Ausführungswegen:

```
private void cmdAnzeige3_Click(...)
{
    int x = (int) numX.Value;
    if (x > 0)
    {
        lblAnzeige.Text = "x ist größer als 0";
        numX.BackColor = Color.LightGreen;
    }
    else
    {
        numX.BackColor = Color.LightBlue;

        if (x < 0)
            lblAnzeige.Text = "x ist kleiner als 0";
```

```
        else
            lblAnzeige.Text = "x ist gleich 0";
    }
}
```

Listing 2.14 Projekt »IfElse«, Teil 3

Zur Erläuterung:

Geschachtelt

- Falls der Wert von x jetzt kleiner oder gleich 0 ist, wird zunächst das Zahlenauswahlfeld hellblau eingefärbt. Außerdem wird eine weitere Untersuchung durchgeführt, da es noch zwei Möglichkeiten gibt.
- Falls der Wert kleiner als 0 ist, erscheint die entsprechende Meldung.
- Falls das nicht der Fall ist, kann der Wert nur noch gleich 0 sein, da vorher alle anderen Fälle ausgeschlossen wurden (siehe Abbildung 2.26).
- Da im sogenannten inneren `if...else` jeweils nur eine Anweisung ausgeführt wird, können hier die geschweiften Klammern weggelassen werden.

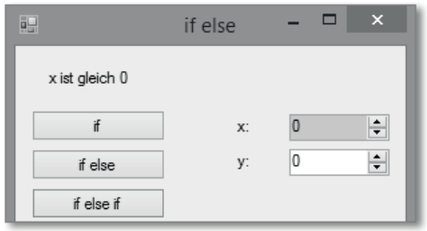


Abbildung 2.26 Drei Möglichkeiten

Es folgt ein Beispiel mit dem logischen Und-Operator &&:

```
private void cmdAnzeige4_Click(...)
{
    int x = (int) numX.Value;
    int y = (int) numY.Value;
    numX.BackColor = Color.White;

    if (x > 0 && y > 0)
        lblAnzeige.Text = "x und y sind größer als 0";
    else
```



```
        lblAnzeige.Text = "Mind. eine der beiden" +
            " Zahlen ist nicht größer als 0";
}
```

Listing 2.15 Projekt »IfElse«, Teil 4

Zur Erläuterung:

- Nun werden beide Zahlenauswahlfelder ausgewertet.
- Falls beide Werte größer als 0 sind, wird der erste Text angezeigt.

Logisches Und

- Falls einer der beiden Werte kleiner oder gleich 0 ist, wird der zweite Text angezeigt (siehe Abbildung 2.27).
- Es wird jeweils nur eine Anweisung ausgeführt, also können die geschweiften Klammern weggelassen werden.

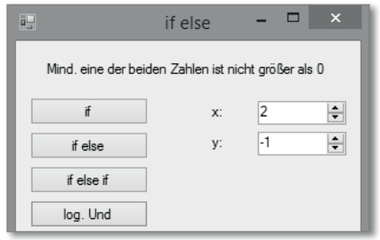


Abbildung 2.27 Logisches Und

Der logische Oder-Operator || liefert ein anderes Ergebnis:

```
private void cmdAnzeige5_Click(...)
{
    int x = (int)numX.Value;
    int y = (int)numY.Value;
    numX.BackColor = Color.White;

    if (x > 0 || y > 0)
        lblAnzeige.Text = "x oder y oder beide" +
            " sind größer als 0";
    else
        lblAnzeige.Text = "Keine der beiden" +
            " Zahlen ist größer als 0";
}
```

Listing 2.16 Projekt »IfElse«, Teil 5

Zur Erläuterung:

Logisches Oder

- Falls einer der Werte oder beide Werte größer als 0 sind, wird der erste Text angezeigt (siehe Abbildung 2.28).

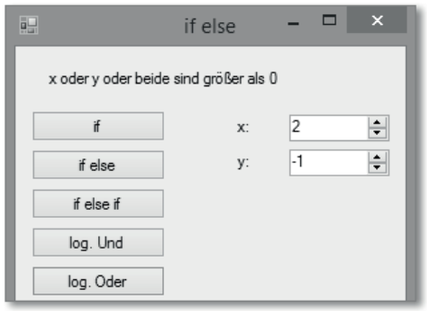


Abbildung 2.28 Logisches Oder

- Falls beide Werte kleiner oder gleich 0 sind, wird der zweite Text angezeigt.

Einen Unterschied zum »normalen« Oder bildet der Exklusiv-Oder-Operator ^:

```
private void cmdAnzeige6_Click(...)
{
    int x = (int)numX.Value;
    int y = (int)numY.Value;
    numX.BackColor = Color.White;
    lblAnzeige.Text = "";
    if (x > 0 ^ y > 0)
        lblAnzeige.Text = "Nur x oder nur y" +
            " ist größer als 0";
}
```

Listing 2.17 Projekt »IfElse«, Teil 6

Zur Erläuterung:

Logisches Exklusiv-Oder

- Es wird etwas angezeigt, falls nur `x` oder nur `y` größer als 0 ist (siehe Abbildung 2.29).
- Falls beide Werte kleiner oder gleich 0 sind oder beide Werte größer als 0 sind, wird nichts angezeigt.

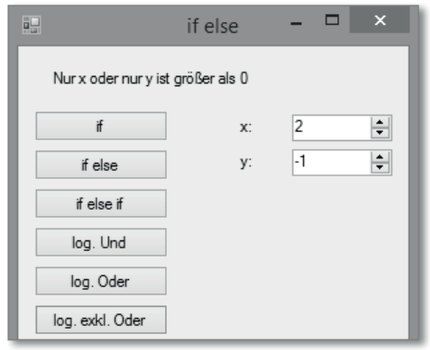


Abbildung 2.29 Logisches Exklusiv-Oder

2.4.2 switch...case

Mehrfachauswahl

Eine Verzweigung kann in bestimmten Fällen auch mit `switch...case` gebildet werden. Diese Struktur vereinfacht eine Mehrfachauswahl, wenn nur ein Wert untersucht werden muss, und ist wie folgt aufgebaut:

```
switch (Testausdruck)
{
   [ case Möglichkeit1:
      Anweisungen1
      [break | goto case MöglichkeitX] ]
   [ case Möglichkeit2:
      Anweisungen2
      [break | goto case MöglichkeitX] ]
   ...
   [ default:
      Anweisungen
      break | goto case MöglichkeitX]
}
```

Testausdruck

Die Struktur `switch...case` verwendet einen Testausdruck, der am Beginn des Blocks ausgewertet wird. Sein Wert wird anschließend der Reihe nach mit den gegebenen Möglichkeiten verglichen. Der Testausdruck kann z. B. eine ganze Zahl, ein einzelnes Zeichen oder eine Zeichenkette sein, aber keine Zahl mit Nachkommastellen.

Alle Anweisungen

Bei der ersten Übereinstimmung einer Möglichkeit mit dem Testausdruck werden die zugehörigen Anweisungen bis zum nächsten `break` oder `goto case` ausgeführt.

break, goto case

Beim Erreichen eines `break` fährt das Programm mit der ersten Anweisung nach dem `switch`-Block fort. Beim Erreichen eines `goto case` fährt das Programm mit der ersten Anweisung der betreffenden Möglichkeit fort.

default

Die `default`-Möglichkeit am Ende des Blocks ist optional. Die zugehörigen Anweisungen werden ausgeführt, falls keine der Möglichkeiten vorher zutraf.

Im nachfolgenden Programm im Projekt *SwitchCase* werden zwei verschiedene Einsatzmöglichkeiten gezeigt. Im ersten Teil wird eine ganze Zahl, die aus einem Zahlenauswahlfeld stammt, untersucht. Es wird festgestellt, ob sie einstellig ungerade, einstellig gerade oder zweistellig ist:

```
private void cmdAnzeigen1_Click_1(...)
{
    int x = (int) numX.Value;

    switch (x)
    {
        case 1:
        case 3:
        case 5:
        case 7:
        case 9:
            lblA.Text = "einstellig, ungerade";
            break;
        case 2:
        case 4:
        case 6:
        case 8:
            lblA.Text = "einstellig, gerade";
            break;
        default:
            lblA.Text = "zweistellig";
            break;
    }
}
```

Listing 2.18 Projekt »SwitchCase«, Teil 1

Zur Erläuterung:

- Wurde eine der Zahlen 1, 3, 5, 7, 9 ausgewählt, trifft eine der ersten fünf Möglichkeiten zu, und es wird *einstellig*, *ungerade* ausgegeben. Erst dann beendet ein `break` den Ablauf innerhalb des `switch`-Blocks.
- Die Zahlen 2, 4, 6 oder 8 führen zur Ausgabe von *einstellig*, *gerade*. Dann folgt ebenfalls ein `break`.
- Es gibt einen `default`-Fall. Falls keine einstellige Zahl ausgewählt wurde, wird *zweistellig* ausgegeben.
- Auf diese Weise führt eine Reihe zusammengehöriger Fälle zu einem gemeinsamen Ausführungsweg.

Im zweiten Teil wird eine gegebene Zeichenkette untersucht:

```
private void cmdAnzeigen2_Click(...)
{
    string s = "Nizza";
    lblA.Text = "";

    switch (s)
    {
        case "France":
            lblA.Text += "Frankreich\n";
            break;
        case "Bordeaux":
            lblA.Text += "Atlantik\n";
            goto case "France";
        case "Nizza":
            lblA.Text += "Cote d'Azur\n";
            goto case "France";
        default:
            lblA.Text += "restliche Fälle\n";
            break;
    }
}
```

Listing 2.19 Projekt »SwitchCase«, Teil 2

Zur Erläuterung:

- Der gegebene Wert der Zeichenkette ist `Nizza`. Es wird *Cote d'Azur* und *Frankreich* ausgegeben, da es nach der ersten Anweisung mit einem `goto case` zum Fall `France` weitergeht.
- Falls der gegebene Wert `France` ist, wird nur *Frankreich* ausgegeben.
- Falls der gegebene Wert `Bordeaux` ist, wird *Atlantik* und *Frankreich* ausgegeben, wiederum wegen eines `goto case`.
- Bei anderen Werten wird der Text *restliche Fälle* ausgegeben.
- Auch auf diese Weise lassen sich Fälle teilweise zusammenführen.

2.4.3 Übungen

Übung ÜSteuerbetrag

Übung ÜSteuerbetrag

Schreiben Sie ein Programm, das zu einem eingegebenen Gehalt den Steuerbetrag berechnet und ausgibt, siehe Abbildung 2.30. In Tabelle 2.10 sind die Steuersätze angegeben. Es wird davon ausgegangen, dass das gesamte Gehalt zum angegebenen Satz versteuert wird.

Gehalt	Steuersatz
bis einschl. 12.000 €	12 %
von 12.000 bis einschl. 20.000 €	15 %
von 20.000 bis einschl. 30.000 €	20 %
über 30.000 €	25 %

Tabelle 2.10 Übung ÜSteuerbetrag

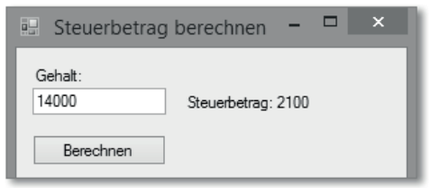


Abbildung 2.30 Übung ÜSteuerbetrag

Übung ÜKranVerzweigung

Übung ÜKran-Verzweigung

Erweitern Sie die Übung *ÜKran* aus Abschnitt 2.3.2, »Zeitgeber«. Die Bewegung des Krans soll kontrolliert werden. Kein Teil des Krans darf zu groß oder zu klein werden. Der Kran darf sich nicht über die sinnvollen Begrenzungen hinaus bewegen. Nutzen Sie Bedingungen und Verzweigungen, um das zu verhindern.

2.5 Verzweigungen und Steuerelemente

In diesem Abschnitt werden Kontrollkästchen und Optionsschaltflächen bzw. Gruppen von Optionsschaltflächen eingeführt. Damit können Zustände unterschieden bzw. Eigenschaften eingestellt werden. Dazu werden Verzweigungen benötigt, die Gegenstand des vorigen Abschnitts 2.4, »Verzweigungen«, waren.

2.5.1 Kontrollkästchen

CheckBox

Das Kontrollkästchen (CheckBox) bietet dem Benutzer die Möglichkeit, zwischen zwei Zuständen zu wählen, z. B. *an* oder *aus*, wie bei einem Schalter. Man kann damit auch kennzeichnen, ob man eine bestimmte optionale Erweiterung wünscht oder nicht. Der Benutzer bedient ein Kontrollkästchen, indem er ein Häkchen setzt oder entfernt.

CheckedChanged

Das wichtigste Ereignis ist beim Kontrollkästchen nicht der `Click`, sondern das Ereignis `CheckedChanged`. Dieses Ereignis zeigt nicht nur an, dass das Kontrollkästchen vom Benutzer bedient wurde, sondern auch, dass es seinen Zustand geändert hat. Das kann beispielsweise auch durch Programmcode geschehen. Eine Ereignismethode zu `CheckedChanged` löst in jedem Fall etwas aus, sobald das Kontrollkästchen (vom Benutzer oder vom Programmcode) geändert wurde.

An/Aus

Allerdings wird der Programmablauf meist so gestaltet, dass bei einem anderen Ereignis der aktuelle Zustand des Kontrollkästchens (an/aus) abgefragt und anschließend entsprechend reagiert wird.

Die wichtigen Eigenschaften des Kontrollkästchens sind:

Checked

- `Checked`: der Zustand der CheckBox mit den Werten `true` und `false`
- `Text`: die Beschriftung neben dem Kontrollkästchen

Im Projekt *Kontrollkästchen* werden alle oben genannten Möglichkeiten genutzt (siehe Abbildung 2.31).

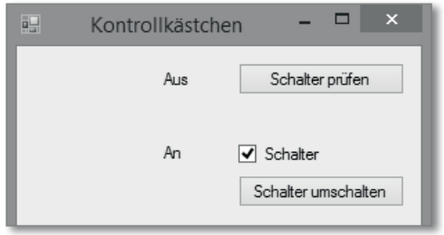


Abbildung 2.31 Zustand nach Klick auf das Kontrollkästchen

Der Programmcode:

```
private void cmdPrüfen_Click(...)
{
    if (chkSchalter.Checked)
        lblTest1.Text = "An";
    else
        lblTest1.Text = "Aus";
}

private void chkSchalter_CheckedChanged(...)
{
    if (chkSchalter.Checked)
        lblTest2.Text = "An";
    else
        lblTest2.Text = "Aus";
}

private void cmdUmschalten_Click(...)
{
    chkSchalter.Checked = !chkSchalter.Checked;
}
```

Listing 2.20 Projekt »Kontrollkästchen«

Zur Erläuterung:

- Der Zustand eines Kontrollkästchens (Häkchen gesetzt oder nicht) kann im Programm mithilfe einer einfachen Verzweigung ausgewertet werden.

Wahrheitswert

- Normalerweise werden bei einer Bedingung in einer Verzweigung zwei Werte durch Vergleichsoperatoren miteinander verglichen, und eines der beiden Ergebnisse `true` oder `false` wird ermittelt. Da die Eigenschaft `Checked` aber bereits einem solchen Wahrheitswert entspricht, kann die Bedingung auch verkürzt formuliert werden. `if (chkSchalter.Checked == true)` hätte also das gleiche Ergebnis erzeugt.
- Die Methode `cmdPrüfen_Click()` wird aufgerufen, wenn der Benutzer den Button SCHALTER PRÜFEN betätigt. Erst in diesem Moment wird der Zustand des Kontrollkästchens (Eigenschaft `Checked` gleich `true` oder `false`) abgefragt und im ersten Label ausgegeben. Es kann also sein, dass das Kontrollkästchen vor längerer Zeit oder noch nie benutzt wurde.
- Dagegen wird die Methode `chkSchalter_CheckedChanged()` sofort aufgerufen, wenn der Benutzer das Kontrollkästchen betätigt, also ein Häkchen setzt oder entfernt. Die Methode wird auch dann aufgerufen, wenn der Benutzer den Zustand des Kontrollkästchens durch Programmcode ändert. Hier wird der Zustand des Kontrollkästchens also unmittelbar nach der Änderung ausgegeben (im zweiten Label).

Umschalten mit !

- Die Methode `cmdUmschalten_Click()` dient dem Umschalten des Kontrollkästchens per Programmcode. Das kommt in Windows-Anwendungen häufig vor, wenn es logische Zusammenhänge zwischen mehreren Steuerelementen gibt. Die Eigenschaft `Checked` wird mithilfe des logischen Operators `!` auf `true` bzw. auf `false` gesetzt. Das führt wiederum zum Ereignis `chkSchalter_CheckedChanged` und dem Ablauf der zugehörigen, oben erläuterten Ereignismethode.

2.5.2 Optionsschaltflächen

RadioButton

Optionsschaltflächen (RadioButtons) treten immer in Gruppen auf und bieten dem Benutzer zwei oder auch mehrere Möglichkeiten zu wählen, etwa zwischen den Farben Rot, Grün und Blau. Bei zusammengehörigen Optionsschaltflächen kann der Benutzer genau eine per Klick auswählen. Alle anderen werden dann unmittelbar als *Nicht ausgewählt* gekennzeichnet.

CheckedChanged

Analog zum Kontrollkästchen ist das wichtigste Ereignis bei einer Optionsschaltfläche `CheckedChanged`. Dieses Ereignis zeigt an, dass die betreffende Optionsschaltfläche ihren Zustand geändert hat. Das kann auch durch Programmcode geschehen.

Der Programmablauf wird hier meist so gestaltet, dass bei einem anderen Ereignis die aktuelle Auswahl innerhalb der Gruppe abgefragt und anschließend je nach Zustand unterschiedlich reagiert wird.

Standardwert

Es ist guter Programmierstil und verringert Folgefehler, wenn Sie eine der Optionsschaltflächen der Gruppe bereits zur Entwicklungszeit auf `true` setzen. Das muss nicht notwendigerweise die erste Optionsschaltfläche der Gruppe sein.

Checked

Die wichtigen Eigenschaften der Optionsschaltflächen sind `Checked` (mit den Werten `true` und `false`) und `Text` (zur Beschriftung). Im nachfolgenden Programm im Projekt *Optionen* werden alle genannten Möglichkeiten genutzt. Es wird der Zustand angezeigt, nachdem der Benutzer

- Blau gewählt,
- den Button PRÜFEN betätigt und
- Grün gewählt hat (siehe Abbildung 2.32).

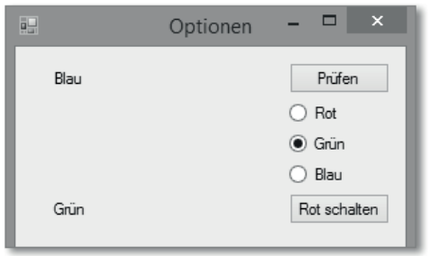


Abbildung 2.32 Zustand nach den genannten Aktionen

Der Programmcode:

```
private void cmdPrüfen_Click(...)
{
    if (optFarbeRot.Checked)
        lblAnzeige1.Text = "Rot";
    else if (optFarbeGrün.Checked)
        lblAnzeige1.Text = "Grün";
    else
        lblAnzeige1.Text = "Blau";
}
```



```
private void optFarbeRot_CheckedChanged(...)
{
    if (optFarbeRot.Checked)
        lblAnzeige2.Text = "Rot";
}

private void optFarbeGrün_CheckedChanged(...)
{
    if (optFarbeGrün.Checked)
        lblAnzeige2.Text = "Grün";
}

private void optFarbeBlau_CheckedChanged(...)
{
    if (optFarbeBlau.Checked)
        lblAnzeige2.Text = "Blau";
}

private void cmdSchalter_Click(...)
{
    optFarbeRot.Checked = true;
}
```

Listing 2.21 Projekt »Optionen«

Zur Erläuterung:

Auswahl oder Abwahl

- Der Zustand einer einzelnen Optionsschaltfläche kann im Programm mithilfe einer einfachen Verzweigung ausgewertet werden. Es muss festgestellt werden, ob diese Optionsschaltfläche ausgewählt oder abgewählt wurde. In beiden Fällen tritt das Ereignis `CheckedChanged` auf.

Mehrfache Verzweigung

- Der Zustand einer Gruppe von Optionsschaltflächen kann im Programm mithilfe einer mehrfachen Verzweigung ausgewertet werden.
- Die Methode `cmdPrüfen_Click()` wird aufgerufen, wenn der Benutzer den Button PRÜFEN betätigt. Erst in diesem Moment wird der Zustand der Gruppe abgefragt und im ersten Label ausgegeben.
- Dagegen wird eine der Methoden `optFarbeRot_CheckedChanged()` (bzw. `...Grün...` oder `...Blau...`) aufgerufen, wenn der Benutzer eine der Optionsschaltflächen auswählt. Diese Methoden werden auch dann aufgerufen, wenn der Benutzer den Zustand der zugehörigen Options-

schaltfläche durch Programmcode ändert. Hier wird der Zustand der Gruppe also unmittelbar nach der Änderung ausgegeben (im zweiten Label).

- Die Methode `cmdSchalter_Click()` dient der Auswahl einer bestimmten Optionsschaltfläche per Programmcode. Das kommt in Windows-Anwendungen häufig vor, wenn es logische Zusammenhänge zwischen mehreren Steuerelementen gibt. Die Eigenschaft `Checked` wird auf `true` gesetzt. Das führt wiederum zum Ereignis `CheckedChanged` der jeweiligen Optionsschaltfläche und zum Ablauf der zugehörigen, oben erläuterten Ereignismethode.

Innerhalb eines Formulars oder einer GroupBox (siehe Abschnitt 2.5.4, »Mehrere Gruppen von Optionsschaltflächen«) kann immer nur bei einer Optionsschaltfläche die Eigenschaft `Checked` den Wert `true` haben. Sobald eine andere Optionsschaltfläche angeklickt wird, ändert sich der Wert der Eigenschaft bei der bisher gültigen Optionsschaltfläche.

2.5.3 Mehrere Ereignisse in einer Methode behandeln

Im folgenden Projekt *MehrereEreignisse* wird eine häufig verwendete Technik vorgestellt. Gibt es mehrere Ereignisse, die auf die gleiche oder auf ähnliche Weise behandelt werden sollen, ist es vorteilhaft, diese Ereignisse mit einer gemeinsamen Ereignismethode aufzurufen.

Dazu gibt es zwei Möglichkeiten:

Methode erzeugen

- Erste Möglichkeit, Teil 1: Sie erzeugen zunächst eine Ereignismethode für das erste Steuerelement auf die gewohnte Art und Weise, nämlich per Doppelklick auf das Steuerelement. Diese Ereignismethode beinhaltet dann namentlich das erste Steuerelement.

Methode zweimal nutzen

- Erste Möglichkeit, Teil 2: Sie markieren das zweite Steuerelement und schalten im EIGENSCHAFTEN-Fenster auf die Ansicht EREIGNISSE (Blitzsymbol) um. Sie gehen in die Zeile mit dem betreffenden Ereignis, klappen auf der rechten Seite eine Liste auf und wählen darin die soeben erzeugte Ereignismethode aus (siehe Abbildung 2.33). Für alle weiteren Steuerelemente, denen dieselbe Ereignismethode zugeordnet werden soll, gehen Sie analog vor.

Neuer Methodenname

- Zweite Möglichkeit, Teil 1: Sie markieren das erste Steuerelement und schalten im EIGENSCHAFTEN-Fenster auf die Ansicht EREIGNISSE um.

Sie gehen in die Zeile mit dem betreffenden Ereignis, tragen darin einen *neutralen* Methodennamen ein (siehe Abbildung 2.33), der für alle betroffenen Steuerelemente passend ist, und betätigen die ↵-Taste. Im Codefenster erscheint die Methode mit dem neutralen Namen.

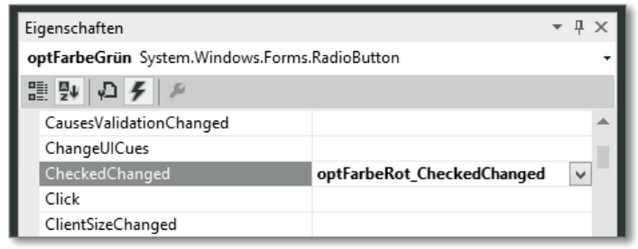


Abbildung 2.33 Auswahl einer vorhandenen Ereignisprozedur

- Zweite Möglichkeit, Teil 2: Für das zweite Steuerelement (und alle weiteren) gehen Sie genau so vor wie bei der ersten Möglichkeit.

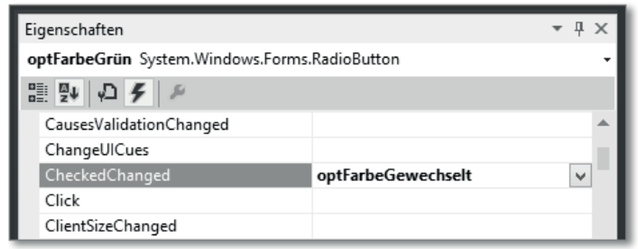


Abbildung 2.34 Eintrag eines eigenen Methodennamens

Unterhalb der Ereignisliste steht eine Erläuterung zu dem jeweiligen Ereignis (siehe Abbildung 2.35).

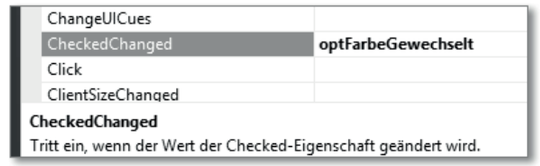


Abbildung 2.35 Erläuterung zum Ereignis

Im nachfolgenden Programm wurde die zweite Möglichkeit mit dem neutralen Namen `optFarbeGewechselt()` verwendet. Der Zustand einer Gruppe von Optionsschaltflächen wird sofort angezeigt, wenn der Benutzer eine davon auswählt:

```
private void optFarbeGewechselt(...)
{
    if (optFarbeRot.Checked)
        lblAnzeige.Text = "Rot";
    else if (optFarbeGrün.Checked)
        lblAnzeige.Text = "Grün";
    else
        lblAnzeige.Text = "Blau";
}
```

Listing 2.22 Projekt »MehrereEreignisse«

Zur Erläuterung:

- Die Methode `optFarbeGewechselt()` wird durch alle drei `CheckedChanged`-Ereignisse aufgerufen.

2.5.4 Mehrere Gruppen von Optionsschaltflächen

Falls in den beiden letzten Programmen weitere Optionsschaltflächen hinzugefügt wurden, gilt nach wie vor: Nur eine der Optionsschaltflächen ist ausgewählt.

Container

Benötigen Sie aber innerhalb eines Formulars mehrere voneinander unabhängige Gruppen von Optionsschaltflächen, wobei in jeder der Gruppen jeweils nur eine Optionsschaltfläche ausgewählt sein soll, müssen Sie jede Gruppe einzeln in einen Container packen. Ein Formular ist bereits ein Container, wir benötigen also einen weiteren Container.

GroupBox

Als ein solcher Container kann beispielsweise das Steuerelement `Gruppe` (GroupBox) dienen. Mit der Zuweisung der Eigenschaft `Text` der GroupBox geben Sie eine Beschriftung an.

Zuordnung

Falls eine GroupBox markiert ist, wird eine neu erzeugte Optionsschaltfläche dieser GroupBox zugeordnet und reagiert gemeinsam mit den anderen Optionsschaltflächen in dieser GroupBox. Anderenfalls wird sie dem Formular zugeordnet und reagiert gemeinsam mit den anderen Optionsschaltflächen, die im Formular außerhalb von GroupBoxen stehen. Sie können eine bereits erzeugte Optionsschaltfläche auch im Nachhinein ausschneiden, das Ziel markieren und sie wieder einfügen, um die Zuordnung zu ändern.

Im Projekt *Optionsgruppen* werden zwei voneinander unabhängige Gruppen von Optionen verwendet (siehe Abbildung 2.36).

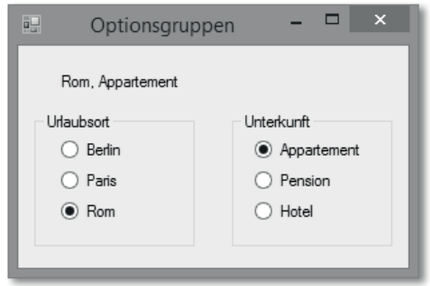


Abbildung 2.36 Zwei Gruppen von RadioButtons

Der Programmcode:

```
public partial class Form1 : Form
{
    ...
    string AusgabeUrlaubsort = "Berlin";
    string AusgabeUnterkunft = "Pension";

    private void optUrlaubsort_CheckedChanged(...)
    {
        // Urlaubsort
        if (optBerlin.Checked)
            AusgabeUrlaubsort = "Berlin";
        else if (optParis.Checked)
            AusgabeUrlaubsort = "Paris";
        else
            AusgabeUrlaubsort = "Rom";

        lblAnzeige.Text = AusgabeUrlaubsort +
            ", " + AusgabeUnterkunft;
    }
```

```
    private void optUnterkunft_CheckedChanged(...)
    {
        // Unterkunft
        if (optAppartement.Checked)
            AusgabeUnterkunft = "Appartement";
        else if (optPension.Checked)
            AusgabeUnterkunft = "Pension";
        else
            AusgabeUnterkunft = "Hotel";

        lblAnzeige.Text = AusgabeUrlaubsort +
            ", " + AusgabeUnterkunft;
    }
}
```

Listing 2.23 Projekt »Optionsgruppen«

Zur Erläuterung:

- Bei einer Urlaubsbuchung können Zielort und Art der Unterkunft unabhängig voneinander gewählt werden. Es gibt also zwei Gruppen von Optionsschaltflächen, jede in einer eigenen GroupBox.
- Bei Auswahl einer der drei Optionsschaltflächen in einer Gruppe wird jeweils die gleiche Methode aufgerufen. In den Methoden wird den klassenweit gültigen Variablen `AusgabeUrlaubsort` und `AusgabeUnterkunft` ein Wert zugewiesen. Anschließend werden die beiden Variablen ausgegeben.
- Die Variablen mussten klassenweit gültig deklariert werden, damit sie in der jeweils anderen Methode zur Verfügung stehen.

Übung

Übung ÜKranOptionen

Erweitern Sie die Übung *ÜKranVerzweigung* aus Abschnitt 2.4, »Verzweigungen«. Die Bewegung des Krans soll per Zeitgeber (Timer) gesteuert werden. Der Benutzer wählt zunächst über eine Gruppe von Optionsschaltflächen aus, welche Bewegung der Kran ausführen soll. Anschließend betätigt er den START-Button (siehe Abbildung 2.37). Die Bewegung wird so lange ausgeführt, bis er den STOP-Button drückt oder eine Begrenzung erreicht wurde.

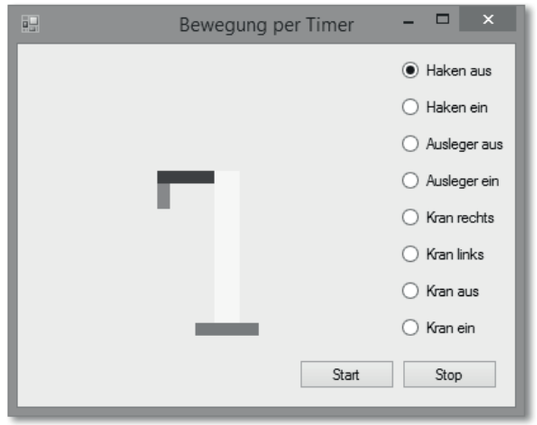


Abbildung 2.37 Übung ÜKranOptionen

2.5.5 Methode ohne Ereignis, Modularisierung

Allgemeine Methode

Bisher wurden nur Methoden behandelt, die mit einem Ereignis zusammenhingen. Darüber hinaus können Sie aber auch unabhängige, allgemeine Methoden schreiben, die von anderen Stellen des Programms aus aufgerufen werden. Diese Methoden können Sie direkt im Codefenster eingeben.

Nachfolgend das Programm im Projekt *MethodeOhneEreignis*, es handelt sich dabei um eine geänderte Version des Programms im Projekt *Optionsgruppen*:

```
public partial class Form1 : Form
{
    ...
    private void optUnterkunft(...)
    {
        // Unterkunft
        if (optAppartement.Checked)
            AusgabeUnterkunft = "Appartement";
        else if (optPension.Checked)
            AusgabeUnterkunft = "Pension";
        else
            AusgabeUnterkunft = "Hotel";

        Anzeigen();
```

```
    }

    private void Anzeigen()
    {
        lblAnzeige.Text = AusgabeUrlaubsort +
            ", " + AusgabeUnterkunft;
    }
}
```

Listing 2.24 Projekt »MethodeOhneEreignis«

Zur Erläuterung:

- Abgebildet wird nur der zweite Teil der Klasse.
- Am Ende der beiden Ereignismethoden `optUnterkunft_CheckedChanged()` und `optUrlaubsort_CheckedChanged()` steht jeweils die Anweisung `Anzeigen()`. Dabei handelt es sich um einen Aufruf der Methode `Anzeigen()`.
- Diese Methode steht weiter unten. Sie ist nicht direkt an ein Ereignis gekoppelt.

Modularisierung

Vorteil dieser Vorgehensweise: Gemeinsam genutzte Programmteile können ausgelagert und müssen nur einmal geschrieben werden. Man nennt diesen Vorgang bei der Programmierung auch Modularisierung. In Abschnitt 4.7, »Methoden«, wird dieses Thema noch genauer behandelt.

2.6 Schleifen

Schleifen werden in Programmen häufig benötigt. Sie ermöglichen den mehrfachen Durchlauf von Anweisungen. Darin liegt eine besondere Stärke der Programmierung allgemein: die schnelle wiederholte Bearbeitung ähnlicher Vorgänge.

Es gibt die Schleifenstrukturen `for`, `while`, `do...while` und `foreach...in`. Mithilfe der Strukturen steuern Sie die Wiederholungen eines Anweisungsblocks (die Anzahl der Schleifendurchläufe). Dabei wird der Wahrheitswert eines Ausdrucks (der Schleifenbedingung) oder der Wert eines numerischen Ausdrucks (Wert des Schleifenzählers) benötigt.

Collection

Die Schleife `foreach...in` wird meist bei Feldern oder Collections (Auflistungen) eingesetzt, siehe Abschnitt 4.6, »foreach-Schleife«.

2.6.1 for-Schleife

for

Falls die Anzahl der Schleifendurchläufe bekannt oder vor Beginn der Schleife berechenbar ist, sollten Sie die `for`-Schleife verwenden. Ihr Aufbau sieht wie folgt aus:

```
for (Startausdruck; Laufbedingung; Änderung)
{
   Anweisungen
   [ break ]
   [ continue ]
}
```

Zur Erläuterung:

- Es wird eine *Schleifenvariable* benutzt, die den Ablauf der Schleife steuert.
- Im *Startausdruck* wird der Startwert der Schleifenvariablen gesetzt.
- Die Schleife läuft, solange die *Laufbedingung* wahr ist. Sie wird im Allgemeinen mit einem Vergleichsoperator gebildet.
- Nach jedem Durchlauf der Schleife wird die Schleifenvariable geändert.

break, continue

Das Schlüsselwort `break` kann eingesetzt werden, um die Schleife aufgrund einer speziellen Bedingung sofort zu verlassen. Das Schlüsselwort `continue` kann eingesetzt werden, um den nächsten Durchlauf der Schleife unmittelbar zu beginnen, ohne den aktuellen Durchlauf zu beenden.

Ohne Klammern

Falls es sich bei `Anweisungen` nur um eine einzelne Anweisung handelt, können die geschweiften Klammern weggelassen werden.

In dem folgenden Programm im Projekt *ForSchleife* werden durch Aufruf von fünf Buttons fünf unterschiedliche Schleifen durchlaufen (siehe Abbildung 2.38).

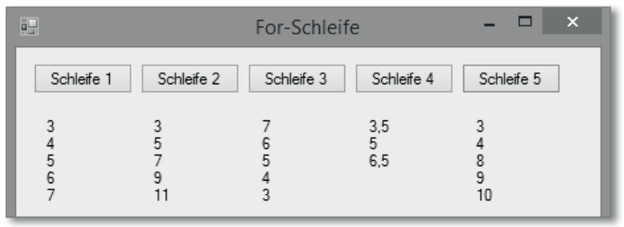


Abbildung 2.38 Verschiedene for-Schleifen

Der Programmcode:

```
private void cmdSchleife1_Click(...)
{
    int i;
    lblA1.Text = "";

    for (i = 3; i <= 7; i++)
    {
        lblA1.Text += i + "\n";
    }
}

private void cmdSchleife2_Click(...)
{
    int i;
    lblA2.Text = "";

    for (i = 3; i <= 11; i = i + 2)
        lblA2.Text += i + "\n";
}

private void cmdSchleife3_Click(...)
{
    int i;
    lblA3.Text = "";

    for (i = 7; i >= 3; i--)
        lblA3.Text += i + "\n";
}

private void cmdSchleife4_Click(...)
{
    double d;
    lblA4.Text = "";

    for (d = 3.5; d <= 7.5; d = d + 1.5)
        lblA4.Text += d + "\n";
}
```



```
private void cmdSchleife5_Click(...)
{
    int i;
    lblA5.Text = "";

    for (i = 3; i <= 20; i++)
    {
        if (i >= 5 && i <= 7)
            continue;
        if (i >= 11)
            break;
        lblA5.Text += i + "\n";
    }
}
```

Listing 2.25 Projekt »ForSchleife«

Zur Erläuterung der ersten Schleife:

- Als Schleifenvariable dient `i`.
- Die Schleife wird erstmalig mit `i = 3` und letztmalig mit `i = 7` durchlaufen.
- Nach jedem Durchlauf wird `i` um 1 erhöht.

Zur Erläuterung der restlichen Schleifen:

- Die zweite, dritte und vierte Schleife beinhalten jeweils nur eine Anweisung, daher konnten die geschweiften Klammern weggelassen werden. Allerdings ist diese Anweisung zur besseren Lesbarkeit eingerückt.
- Bei der zweiten Schleife wurde die Schrittweite 2 gewählt.
- Die dritte Schleife läuft abwärts, daher muss die Schleifenvariable vermindert werden.
- In der vierten Schleife wird gezeigt, dass eine Schleife auch nicht ganzzahlige Werte durchlaufen kann.

continue

- In der fünften Schleife werden die Werte 5 bis 7 nicht ausgegeben. Das Schlüsselwort `continue` sorgt dafür, dass der Rest der Anweisungen in der Schleife übersprungen und direkt mit dem nächsten Durchlauf fortgefahren wird.

break

- Eigentlich läuft diese fünfte Schleife bis 20. Aufgrund des Schlüsselworts `break` wird sie allerdings vorzeitig beendet.

Endlos-Schleife

Sie sollten darauf achten, dass *Startausdruck*, *Laufbedingung* und *Änderung* so gestaltet werden, dass keine Endlos-Schleife konstruiert wird. Die Schleife `for(i=5; i<=10; i--)` läuft endlos, da `i` kleiner wird und daher die Laufbedingung immer wahr ist.

Eine reine Schleifenvariable kann auch innerhalb des Kopfs der `for`-Schleife deklariert werden. So kann die erste Schleife in diesem Projekt auch wie folgt geschrieben werden:

```
for (int i = 3; i <= 7; i++) { ... }
```

2.6.2 while- und do...while-Schleife

Steuerung über Bedingung

Ist die Anzahl der Schleifendurchläufe nicht bekannt bzw. vor Beginn der Schleife nicht berechenbar, sollten Sie die `while`-Schleife oder die `do...while`-Schleife verwenden.

Das ist der Aufbau der `while`-Schleife:

```
while (Laufbedingung)
{
   Anweisungen
   [ break ]
   [ continue ]
}
```

Es folgt der Aufbau der `do...while`-Schleife:

```
do
{
   Anweisungen
   [ break ]
   [ continue ]
}
while (Laufbedingung)
```

Zur Erläuterung:

- Die Schleifen werden durchlaufen, solange die Laufbedingung wahr ist.
- Der Unterschied: Die `do...while`-Schleife wird mindestens einmal durchlaufen, da die Laufbedingung erst am Ende geprüft wird.

Ohne Klammern

Falls es sich bei `Anweisungen` nur um eine einzelne Anweisung handelt, können Sie die geschweiften Klammern weglassen.

Die Schlüsselwörter `break` und `continue` haben die gleiche Wirkung wie bei der `for`-Schleife.

Zufallsgenerator

Im folgenden Programm im Projekt *WhileDoWhileSchleifen* werden Zahlen addiert, solange die Summe der Zahlen kleiner als 20 ist, siehe Abbildung 2.39. Da die Zahlen durch einen Zufallsgenerator erzeugt werden, ist die Anzahl der Schleifendurchläufe nicht vorhersagbar.

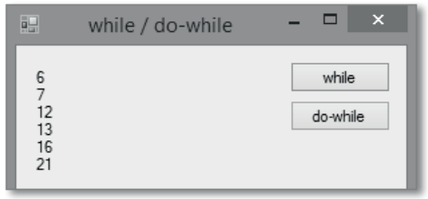


Abbildung 2.39 Bedingungsgesteuerte Schleife

Random, Next()

Der Zufallszahlengenerator wird mithilfe eines Objekts der Klasse `Random` realisiert, das klassenweit gültig deklariert wird. Die Methode `Next()` der Klasse `Random` liefert quasi-zufällige ganze Zahlen. An die Methode `Next()` werden zwei Zahlen in Klammern übergeben. Die erste Zahl steht für die kleinste mögliche Zufallszahl, die zweite Zahl minus 1 kennzeichnet die größte mögliche Zufallszahl.

```
public partial class Form1 : Form
{
    ...
    Random r = new Random();

    private void cmdWhile_Click(...)
    {
        int summe = 0, z;
        lblA.Text = "";

        while (summe < 20)
        {
            z = r.Next(1, 7);
            summe = summe + z;
```

```
            lblA.Text += summe + "\n";
        }
    }

    private void cmdDoWhile_Click(...)
    {
        int summe = 0, z;
        lblA.Text = "";

        do
        {
            z = r.Next(1, 7);
            summe = summe + z;
            lblA.Text += summe + "\n";
        }
        while (summe < 20);
    }
}
```

Listing 2.26 Projekt »WhileDoWhileSchleifen«

while

Zur Erläuterung der `while`-Schleife:

- Die Variable `summe` wird zunächst mit dem Wert 0 initialisiert.
- Zu Beginn der Schleife (kopfgesteuerte Schleife) wird geprüft, ob die Summe der Zahlen kleiner als 20 ist. Trifft das zu, kann die Schleife durchlaufen werden.

Summe berechnen

- Der Wert der Variablen `summe` wird um eine Zufallszahl zwischen 1 und 6 erhöht.
- Der Inhalt des Labels wird um den aktuellen Wert der `summe` und einen Zeilenumbruch verlängert.
- Nach Durchlauf einer Schleife wird das Programm wieder am Beginn der Schleife fortgesetzt. Es wird wiederum geprüft, ob die Summe der Zahlen kleiner als 20 ist.

do...while

Zur Erläuterung der `do...while`-Schleife:

- Die Schleife wird mindestens einmal durchlaufen, selbst wenn die Summe der Zahlen größer oder gleich 20 ist. Für diesen Fall ist also die `do...while`-Schleife nicht so gut geeignet.

- Erst am Ende (fußgesteuerte Schleife) wird geprüft, ob die Summe der Zahlen kleiner als 20 ist. Trifft das zu, wird das Programm wieder am Beginn der Schleife fortgesetzt.

Hinweis: Bei einer `while`-Schleife könnte es vorkommen, dass sie niemals durchlaufen wird.

Endlos-Schleife

Sie sollten wie bei der `for`-Schleife darauf achten, dass keine Endlos-Schleife konstruiert wird. Falls in einer der beiden oben genannten Schleifen die Variable `summe` ihren Wert nicht ändern würde, wäre das eine solche Endlos-Schleife.

2.6.3 Übungen

Anhand einer Reihe von Übungsaufgaben zu Schleifen (und Verzweigungen) trainieren Sie im Folgenden einige typische Probleme der Programmierung in Visual C#. Der visuelle Teil der Lösung enthält in der Regel nur ein einfaches Textfeld zur Eingabe, einen oder zwei Buttons zum Durchführen der Aufgabe und ein einfaches Label zur Ausgabe.

Übung ÜForSchleife, Teil 1

Übung ÜForSchleife, Teil 1

`for`-Schleife: Schreiben Sie ein Programm mit einer einfachen Schleife, das nacheinander die folgenden Zahlen ausgibt: 35; 32,5; 30; 27,5; 25; 22,5; 20.

Übung ÜForSchleife, Teil 2

Übung ÜForSchleife, Teil 2

`for`-Schleife: Erweitern Sie die vorige Aufgabe. Am Ende der Zeile sollen Summe und Mittelwert aller Zahlen angezeigt werden (siehe Abbildung 2.40).

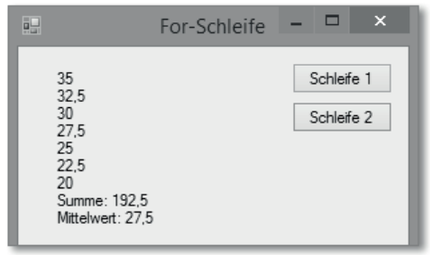


Abbildung 2.40 Übung ÜForSchleife

Übung ÜHalbierung

Übung ÜHalbierung

`while`- oder `do...while`-Schleife: Schreiben Sie ein Programm, mit dessen Hilfe eine eingegebene Zahl wiederholt halbiert und ausgegeben wird. Das Programm soll beendet werden, wenn das Ergebnis der Halbierung kleiner als 0,001 ist (siehe Abbildung 2.41). Falls die Zahl schon zu Beginn kleiner als 0,001 ist, soll sie nicht halbiert werden.

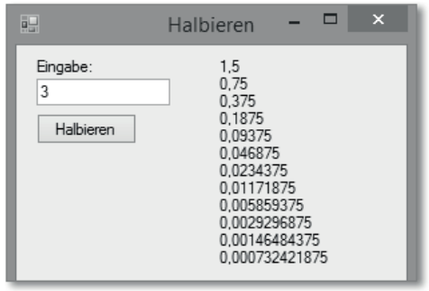


Abbildung 2.41 Übung ÜHalbierung

Übung ÜZahlenraten

Übung ÜZahlenraten

`if...else`: Schreiben Sie ein Programm, mit dem das Spiel *Zahlenraten* gespielt werden kann: Per Zufallsgenerator wird eine Zahl zwischen 1 und 100 erzeugt, aber nicht angezeigt. Der Benutzer soll so lange Zahlen eingeben, bis er die Zahl erraten hat. Als Hilfestellung soll jedes Mal ausgegeben werden, ob die eingegebene Zahl größer oder kleiner als die zu ratende Zahl ist (siehe Abbildung 2.42).

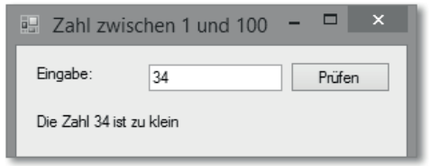


Abbildung 2.42 Übung ÜZahlenraten

Übung ÜSteuertabelle

Übung ÜSteuertabelle

`for`-Schleife und `if...else`: Erweitern Sie das Programm aus der Übung *ÜSteuerbetrag* aus Abschnitt 2.4, »Verzweigungen«. Schreiben Sie ein Programm, das zu einer Reihe von Gehältern u. a. den Steuerbetrag berechnet und ausgibt. In Tabelle 2.11 sind die Steuersätze angegeben.

Gehalt	Steuersatz
bis einschl. 12.000 €	12 %
von 12.000 bis einschl. 20.000 €	15 %
von 20.000 bis einschl. 30.000 €	20 %
über 30.000 €	25 %

Tabelle 2.11 Übung ÜSteuertabelle

Es sollen für jedes Gehalt von 5.000 € bis 35.000 € in Schritten von 3.000 € folgende vier Werte ausgegeben werden: Gehalt, Steuersatz, Steuerbetrag, Gehalt abzüglich Steuerbetrag. Jedes Gehalt soll mit den zugehörigen Werten in einer eigenen Zeile ausgegeben werden (siehe Abbildung 2.43). Auch hier wird davon ausgegangen, dass das gesamte Gehalt zum angegebenen Satz versteuert wird.

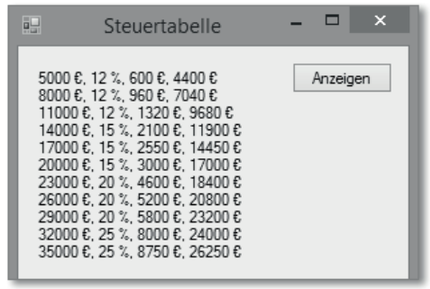


Abbildung 2.43 Übung ÜSteuertabelle

2.7 Schleifen und Steuerelemente

In diesem Abschnitt werden die beiden Steuerelemente *Listenfeld* und *Kombinationsfeld* eingeführt. Damit kann eine einfache oder mehrfache Auswahl aus mehreren Möglichkeiten getroffen werden. Im Zusammenhang mit diesen Steuerelementen werden häufig Schleifen benötigt, wie sie im vorigen Abschnitt behandelt wurden.

2.7.1 Listenfeld

ListBox

Ein Listenfeld (ListBox) zeigt eine Liste mit Einträgen an, aus denen der Benutzer einen oder mehrere auswählen kann. Enthält das Listenfeld mehr Einträge, als gleichzeitig angezeigt werden können, erhält es automatisch einen Scrollbalken.

Items

Die wichtigste Eigenschaft des Steuerelements ListBox ist die Collection `Items`. Sie enthält die einzelnen Listeneinträge. Listenfelder können Sie zur Entwurfszeit füllen, indem Sie der Eigenschaft `Items` in einem eigenen kleinen Dialogfeld die Einträge hinzufügen. In der Regel werden Sie ein Listenfeld aber zur Laufzeit füllen.

2.7.2 Listenfeld füllen

Items.Add()

Bisher wurden die Eigenschaften und Ereignisse von Steuerelementen behandelt. Darüber hinaus gibt es jedoch auch spezifische Methoden, die auf diese Steuerelemente bzw. auf deren Eigenschaften angewendet werden können. Beim Listenfeld ist das u. a. die Methode `Add()` der Eigenschaft `Items`.

Diese Methode nutzen Sie am sinnvollsten einmalig zum Zeitpunkt des Ladens des Formulars. Dieser Zeitpunkt wird durch das Ereignis `Load` gekennzeichnet. Sie erstellen den Rahmen der zugehörigen Ereignismethode, indem Sie einen Doppelklick auf einer freien Stelle des Formulars ausführen. Die Klasse des Formulars heißt, falls Sie das nicht verändern, `Form1`, die Methode hat demnach den Namen `Form1_Load()`.

Im nachfolgenden Programm im Projekt *ListenfeldFüllen* wird ein Listenfeld für italienische Speisen zu Beginn des Programms mit den folgenden Werten gefüllt: *Spaghetti*, *Grüne Nudeln*, *Tortellini*, *Pizza*, *Lasagne* (siehe Abbildung 2.44).

Abbildung 2.44 Listenfeld mit Scrollbalken

Der Programmcode:

```
private void Form1_Load(...)
{
    lstSpeisen.Items.Add("Spaghetti");
    lstSpeisen.Items.Add("Grüne Nudeln");
    lstSpeisen.Items.Add("Tortellini");
    lstSpeisen.Items.Add("Pizza");
    lstSpeisen.Items.Add("Lasagne");
}
```

Listing 2.27 Projekt »ListenfeldFüllen«

Zur Erläuterung:

- Das Ereignis `Load` wird ausgelöst, wenn das Formular geladen wird.
- Die einzelnen Speisen werden der Reihe nach dem Listenfeld hinzugefügt. *Lasagne* steht anschließend ganz unten.

2.7.3 Wichtige Eigenschaften

Die folgenden Eigenschaften eines Listenfelds bzw. der Collection `Items` werden in der Praxis häufig benötigt:

- `Items.Count` gibt die Anzahl der Elemente in der Liste an.
- `SelectedItem` beinhaltet das aktuell vom Benutzer ausgewählte Element der Liste. Wurde kein Element ausgewählt, ergibt `SelectedItem` nichts. **SelectedItem**
- `SelectedIndex` gibt die laufende Nummer des aktuell vom Benutzer ausgewählten Elements an, beginnend bei 0 für das oberste Element. Wurde kein Element ausgewählt, ergibt `SelectedIndex` den Wert –1.
- Über `Items(Index)` können Sie die einzelnen Elemente ansprechen, das oberste Element ist `Items(0)`. **Items[i]**

Das folgende Programm im Projekt *ListenfeldEigenschaften* veranschaulicht alle diese Eigenschaften (siehe Abbildung 2.45).

Der Programmcode:

```
private void cmdAnzeige_Click(...)
{
    int i;
```

```
        lblAnzeige1.Text =
            "Anzahl: " + lstSpeisen.Items.Count;
        lblAnzeige2.Text = "Ausgewählter Eintrag: " +
            lstSpeisen.SelectedItem;
        lblAnzeige3.Text = "Nummer des ausgewählten" +
            " Eintrags: " + lstSpeisen.SelectedIndex;

        lblAnzeige4.Text = "Alle Einträge:" + "\n";
        for (i = 0; i < lstSpeisen.Items.Count; i++)
            lblAnzeige4.Text +=
                lstSpeisen.Items[i] + "\n";
    }
```

Listing 2.28 Projekt »ListenfeldEigenschaften«

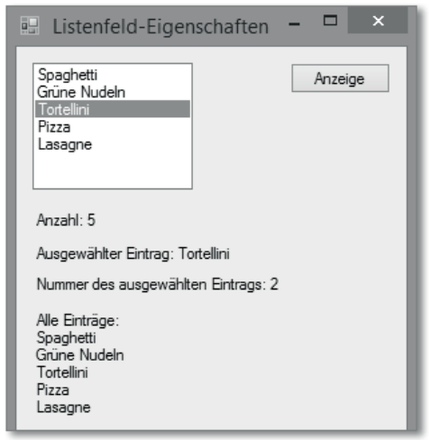


Abbildung 2.45 Anzeige nach Auswahl eines Elements

Zur Erläuterung:

- Das Listenfeld ist bereits gefüllt, siehe Projekt *ListenfeldFüllen*.

Count

- Die Anzahl der Elemente wird über `lstSpeisen.Items.Count` ausgegeben, in diesem Fall sind es fünf.
- Der ausgewählte Eintrag steht in `lstSpeisen.SelectedItem`, seine Nummer in `lstSpeisen.SelectedIndex`.

- Eine `for`-Schleife dient zur Ausgabe aller Elemente. Sie läuft von 0 bis `lstSpeisen.Items.Count – 1`. Das liegt daran, dass bei einer Liste mit fünf Elementen die Elemente mit 0 bis 4 nummeriert sind.

Items[i]

- Die einzelnen Elemente werden mit `lstSpeisen.Items[i]` angesprochen. Die Variable `i` beinhaltet bei der Schleife die aktuelle laufende Nummer.

2.7.4 Wechsel der Auswahl

SelectedIndex-Changed

Ähnlich wie beim Kontrollkästchen oder bei der Optionsschaltfläche ist das wichtigste Ereignis einer ListBox nicht der `Click`, sondern das Ereignis `SelectedIndexChanged`. Dieses Ereignis zeigt nicht nur an, dass die ListBox vom Benutzer bedient wurde, sondern auch, dass sie ihren Zustand geändert hat. Das kann z. B. auch durch Programmcode geschehen. Eine Ereignismethode zu `SelectedIndexChanged` wird in jedem Fall durchlaufen, sobald die ListBox (vom Benutzer oder vom Programmcode) geändert wurde.

Allerdings wird der Programmablauf meist so gestaltet, dass bei einem anderen Ereignis die aktuelle Auswahl der ListBox abgefragt wird und anschließend je nach Zustand unterschiedlich reagiert wird. Das nachfolgende Programm im Projekt *ListenfeldEreignis* veranschaulicht diesen Zusammenhang (siehe Abbildung 2.46).

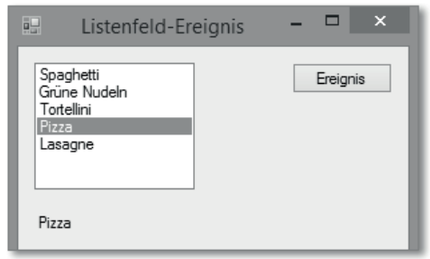


Abbildung 2.46 Anzeige nach dem Ereignis

Der Programmcode:

```
private void cmdEreignis_Click(...)
{
    lstSpeisen.SelectedIndex = 3;
}
```

```
private void lstSpeisen_SelectedIndexChanged(...)
{
    lblAnzeige.Text =
        "Auswahl: " + lstSpeisen.SelectedItem;
}
```

Listing 2.29 Projekt »ListenfeldEreignis«

Zur Erläuterung:

- Das Listenfeld ist bereits gefüllt, siehe Projekt *ListenfeldFüllen*.
- In der Ereignismethode `cmdEreignis_Click()` wird die Nummer des ausgewählten Elements auf 3 gesetzt. Dadurch wird in der ListBox *Pizza* ausgewählt. Im Label wird die geänderte Auswahl sofort angezeigt, da das Ereignis `lstSpeisen_SelectedIndexChanged` ausgelöst wurde.
- In der zugehörigen Ereignismethode `lstSpeisen_SelectedIndexChanged()` wird die Anzeige des ausgewählten Elements ausgelöst. Dieses wird unmittelbar nach der Auswahl angezeigt. Die Auswahl kann durch einen Klick des Benutzers in der Liste oder auch durch Programmcode ausgelöst werden.

2.7.5 Wichtige Methoden

Die Methoden `Insert()` und `RemoveAt()` können Sie zur Veränderung der Inhalte des Listenfelds nutzen:

Insert()
- Mithilfe der Methode `Insert()` können Sie Elemente zum Listenfeld an einer gewünschten Stelle hinzufügen.

RemoveAt()
- Die Methode `RemoveAt()` löscht ein Element an der gewünschten Stelle.

Im nachfolgenden Programm im Projekt *ListenfeldMethoden* werden die beiden Methoden eingesetzt, um ein Listenfeld zu verwalten (siehe Abbildung 2.47).

Sie können Elemente einfügen, löschen und ändern. Um sicherzustellen, dass es sich hierbei um sinnvolle Operationen handelt, sollten Sie jeweils bestimmte Bedingungen beachten:

```
private void cmdLöschen_Click(...)
{
    int x = lstSpeisen.SelectedIndex;
    if (x != -1)
```

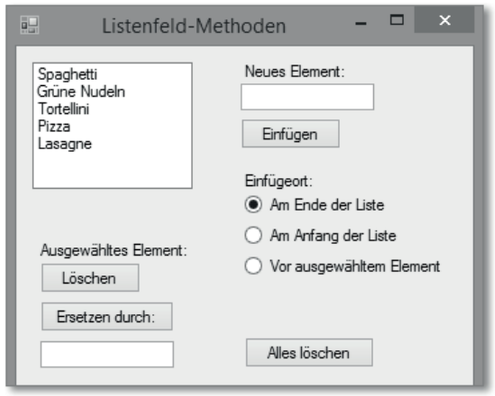



Abbildung 2.47 Verwaltung eines Listenfelds

```
        lstSpeisen.Items.RemoveAt(x);
}

private void cmdEinfügen_Click(...)
{
    if (txtNeu.Text == "")
        return;

    if (optAnfang.Checked)
        lstSpeisen.Items.Insert(0, txtNeu.Text);
    else if (optAuswahl.Checked &&
            lstSpeisen.SelectedIndex != -1)
        lstSpeisen.Items.Insert(
            lstSpeisen.SelectedIndex,
            txtNeu.Text);
    else
        lstSpeisen.Items.Add(txtNeu.Text);
    txtNeu.Text = "";
}

private void cmdErsetzen_Click(...)
{
    int x;

    if (txtErsetzen.Text != "" &&
```

```
            lstSpeisen.SelectedIndex != -1)
        {
            x = lstSpeisen.SelectedIndex;
            lstSpeisen.Items.RemoveAt(x);
            lstSpeisen.Items.Insert(
                x, txtErsetzen.Text);
            txtErsetzen.Text = "";
        }
    }

    private void cmdAllesLöschen_Click(...)
    {
        lstSpeisen.Items.Clear();
    }
```

Listing 2.30 Projekt »ListenfeldMethoden«

Zur Erläuterung:

- Das Listenfeld ist bereits gefüllt, siehe Projekt *ListenfeldFüllen*.
- In der Methode `cmdLöschen_Click()` wird der Wert von `SelectedIndex` in der Variablen `x` gespeichert. Anschließend wird untersucht, ob ein Element ausgewählt wurde, ob also der Wert von `x` ungleich –1 ist. Ist das der Fall, wird dieses Element mit der Methode `RemoveAt()` gelöscht. Wurde kein Element ausgewählt, geschieht nichts.

return

- In der Methode `cmdEinfügen_Click()` wird zunächst die TextBox untersucht. Falls diese leer ist, wird die Methode mit dem Schlüsselwort `return` unmittelbar verlassen. Steht etwas in der TextBox, wird untersucht, welcher Einfügeort über die Optionsschaltflächen ausgesucht wurde:

Add()

  - Wurde als Einfügeort das Ende der Liste gewählt, wird der Inhalt der TextBox mit der bekannten Methode `Add()` am Ende der Liste angefügt.

Insert()

  - In den beiden anderen Fällen wird die Methode `Insert()` zum Einfügen des Inhalts der TextBox vor einem vorhandenen Listeneintrag genutzt. Diese Methode benötigt den Index des Elements, vor dem eingefügt werden soll. Das ist entweder der Wert 0, falls am Anfang der Liste eingefügt werden soll, oder der Wert von `SelectedIndex`, falls vor dem ausgewählten Element eingefügt werden soll.

- Anschließend wird die TextBox gelöscht, damit nicht versehentlich zweimal das gleiche Element eingefügt wird.
- In der Methode `cmdErsetzen_Click()` wird untersucht, ob in der TextBox etwas zum Ersetzen steht und ob ein Element zum Ersetzen ausgewählt wurde. Ist das der Fall, wird
  - der Wert von `SelectedIndex` in der Variablen `x` gespeichert,
  - das zugehörige Element mit der Methode `RemoveAt()` gelöscht,
  - der neue Text an der gleichen Stelle mit der Methode `Insert()` eingefügt und
  - die TextBox gelöscht, damit nicht versehentlich zweimal das gleiche Element eingefügt wird.

RemoveAt()

- In der Methode `cmdAllesLöschen_Click()` dient die Methode `Clear()` zum Leeren der ListBox.

Nach einigen Änderungen sieht das Listenfeld wie das in Abbildung 2.48 aus.

Hinweis: Das Schlüsselwort `return` dient nicht nur dem unmittelbaren Beenden einer Methode, sondern auch dem Liefern des Rückgabewerts einer Methode, siehe Abschnitt 4.7.3, »Methoden mit Rückgabewerten«.

return

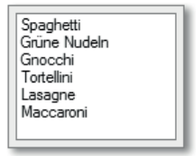


Abbildung 2.48 Nach einigen Änderungen

2.7.6 Mehrfachauswahl

Sie können dem Benutzer ermöglichen, gleichzeitig mehrere Einträge aus einer Liste auszuwählen, wie er es auch aus anderen Windows-Programmen kennt. Dazu wird zur Entwicklungszeit die Eigenschaft `SelectionMode` auf den Wert `MultiExtended` gesetzt. Der Benutzer kann anschließend mithilfe der Strg-Taste mehrere einzelne Elemente auswählen oder mithilfe der ⇧-Taste (wie für Großbuchstaben) einen zusammenhängenden Bereich von Elementen markieren.

SelectionMode

Hinweis: Nach dem Einfügen einer neuen ListBox in ein Formular steht die Eigenschaft `SelectionMode` zunächst auf dem Standardwert `One`, was bedeutet, dass nur ein Element ausgewählt werden kann.

SelectedIndices

Die Eigenschaften `SelectedIndices` und `SelectedItems` beinhalten die Nummern bzw. die Einträge der ausgewählten Elemente. Sie ähneln in ihrem Verhalten der Eigenschaft `Items`. Das nachfolgende Programm im Projekt *ListenfeldMehrfachauswahl* verdeutlicht das (siehe Abbildung 2.49).

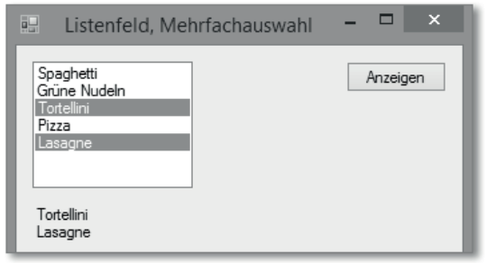


Abbildung 2.49 Mehrere ausgewählte Elemente

Der Programmcode:

```
private void cmdAnzeigen_Click(...)
{
    int i;
    lblAnzeige.Text = "";

    for (i=0; i<lstSpeisen.SelectedItems.Count; i++)
        lblAnzeige.Text +=
            lstSpeisen.SelectedItems[i] + "\n";
}
```

Listing 2.31 Projekt »ListenfeldMehrfachauswahl«

Zur Erläuterung:

SelectedItems[i]

- Das Listenfeld ist bereits gefüllt, siehe Projekt *ListenfeldFüllen*.
- In der Methode `cmdAnzeigen_Click()` werden alle ausgewählten Elemente mithilfe einer Schleife durchlaufen. Diese Schleife läuft von `0` bis `SelectedItems.Count – 1`. Die ausgewählten Elemente selbst werden über `SelectedItems[i]` angesprochen.

2.7.7 Kombinationsfelder

Das Steuerelement *Kombinationsfeld* (ComboBox) vereinigt die Merkmale eines Listenfelds mit denen eines Textfelds. Der Benutzer kann einen Eintrag aus dem Listenfeldbereich auswählen oder ihn in den Textfeldbereich eingeben. Das Kombinationsfeld hat im Wesentlichen die Eigenschaften und Methoden des Listenfelds.

DropDownStyle

Sie können mithilfe der Eigenschaft `DropDownStyle` zwischen drei Typen von Kombinationsfeldern wählen:

- `DropDown`: Das ist die Standardauswahl aus einer Liste (Aufklappen der Liste mit der Pfeiltaste) oder die Eingabe in das Textfeld. Das Kombinationsfeld hat die Größe einer TextBox.
- `DropDownList`: Die Auswahl ist begrenzt auf die Einträge der aufklappbaren Liste, also ohne eigene Eingabemöglichkeit. Dieser Typ Kombinationsfeld verhält sich demnach wie ein Listenfeld, ist allerdings so klein wie eine TextBox. Ein Listenfeld könnte zwar auch auf diese Größe verkleinert werden, aber die Scrollpfeile sind dann sehr klein.
- `Simple`: Die Liste ist immer geöffnet und wird bei Bedarf mit einer Bildlaufleiste versehen. Wie beim Typ `DropDown` ist die Auswahl aus der Liste oder die Eingabe in das Textfeld möglich. Beim Erstellen eines solchen Kombinationsfelds kann die Höhe wie bei einer ListBox eingestellt werden.

Die Eigenschaft `SelectionMode` gibt es bei Kombinationsfeldern nicht. Das folgende Programm im Projekt *Kombinationsfeld* führt alle drei Typen von Kombinationsfeldern vor (siehe Abbildung 2.50).

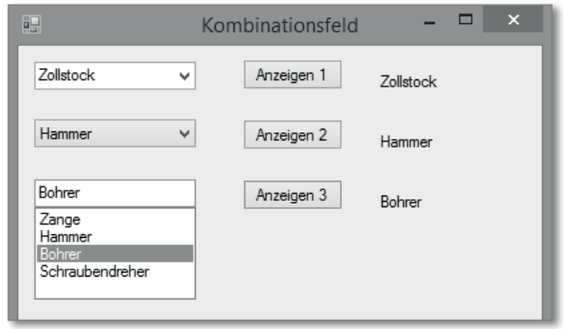


Abbildung 2.50 Drei verschiedene Kombinationsfelder

Der Programmcode:

```
private void Form1_Load(...)
{
    cmbWerkzeug1.Items.Add("Zange");
    cmbWerkzeug1.Items.Add("Hammer");
    cmbWerkzeug1.Items.Add("Bohrer");
    cmbWerkzeug1.Items.Add("Schraubendreher");

[... das Gleiche für die beiden anderen Kombinationsfelder ...]
}

private void cmdAnzeigen1_Click(...)
{
    lblAnzeige1.Text =
        "Auswahl: " + cmbWerkzeug1.Text;
}

private void cmdAnzeigen2_Click(...)
{
    lblAnzeige2.Text =
        "Auswahl: " + cmbWerkzeug2.SelectedItem;
}
private void cmdAnzeigen3_Click(...)
{
    lblAnzeige3.Text =
        "Auswahl: " + cmbWerkzeug3.Text;
}
```

Listing 2.32 Projekt »Kombinationsfeld«

Zur Erläuterung:

- Das erste Kombinationsfeld hat den `DropDownStyle DropDown`. Hat der Benutzer einen Eintrag ausgewählt, erscheint dieser in der TextBox des Kombinationsfelds. Falls er selbst einen Eintrag eingibt, wird dieser ebenfalls dort angezeigt. Die Eigenschaft `Text` enthält den Inhalt dieser TextBox, also immer den Wert des Kombinationsfelds.
- Das zweite Kombinationsfeld hat den `DropDownStyle DropDownList`. Es gibt also keine TextBox. Wie beim Listenfeld ermitteln Sie die Auswahl des Benutzers über die Eigenschaft `SelectedItem`.

- Das dritte Kombinationsfeld hat den `DropDownStyle Simple`. Im Programm kann es genauso wie das erste Kombinationsfeld behandelt werden. Die Eigenschaft `Text` beinhaltet also immer den Wert des Kombinationsfelds.

Übung

Übung ÜListenfeld

Schreiben Sie ein Programm, das zwei Listenfelder beinhaltet, in denen jeweils mehrere Elemente markiert werden können. Zwischen den beiden Listenfeldern befinden sich zwei Buttons, jeweils mit einem Pfeil nach rechts bzw. nach links (siehe Abbildung 2.51). Bei Betätigung eines der beiden Buttons sollen die ausgewählten Elemente in Pfeilrichtung aus der einen Liste in die andere Liste verschoben werden (siehe Abbildung 2.52).

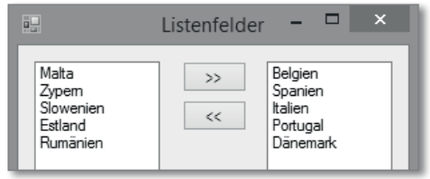


Abbildung 2.51 Liste vor dem Verschieben

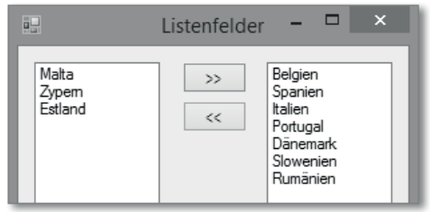


Abbildung 2.52 Liste nach dem Verschieben

SelectedIndices

Bei der Lösung kann neben der Eigenschaft `SelectedItems` z. B. auch die Eigenschaft `SelectedIndices` genutzt werden. Eine solche Collection beinhaltet dann nicht die ausgewählten Einträge, sondern deren Indizes. Mit dem Löschen mehrerer Einträge aus einem Listenfeld sollten Sie vom Ende der Liste her beginnen. Der Grund hierfür ist: Löschen Sie eines der vorderen Elemente zuerst, stimmen die Indizes in der Collection `SelectedIndices` nicht mehr.

Kapitel 11
Beispielprojekte

Als weiterführende Übungsaufgaben werden in diesem Kapitel zwei lauffähige Beispielprojekte vorgeführt. Haben Sie den geschilderten Aufbau verstanden, können Sie später eigene Verbesserungen oder Erweiterungen einbringen.

Bei den beiden Beispielprojekten handelt es sich zum einen um das bekannte Tetris-Spiel und zum anderen um einen Vokabeltrainer.

11.1 Spielprogramm Tetris

Im Folgenden wird das bekannte Spielprogramm Tetris in einer vereinfachten, nachvollziehbaren Version für Visual C# realisiert und erläutert. Das Programm beinhaltet:

- ein zweidimensionales Feld
- einen Timer
- einen Zufallsgenerator
- die Erzeugung und Löschung von Steuerelementen zur Laufzeit
- die Zuordnung von Ereignismethoden zu Steuerelementen, die erst zur Laufzeit erzeugt werden

Abbildung 11.1 zeigt die Benutzeroberfläche des Programms.

11.1.1 Spielablauf

Panel fällt herunter

Nach Programmstart fällt ein Steuerelement vom Typ `Panel` in einer von acht möglichen Farben so weit herunter, bis es auf den Rand des Spielfelds oder auf ein anderes Panel trifft. Es kann mithilfe der drei Buttons »Links« (Li), »Rechts« (Re) und »Drop« (Dr) bewegt werden. »Drop« bewirkt ein sofortiges Absenken des Panels auf die unterste mögliche Position.

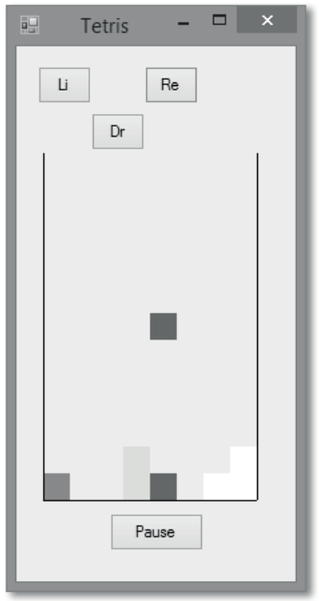


Abbildung 11.1 Tetris

Nächstes Level

Befinden sich drei gleichfarbige Panels untereinander oder nebeneinander, verschwinden sie. Panels, die sich eventuell darüber befinden, rutschen nach. Anschließend wird die Fallgeschwindigkeit der Panels erhöht. Das bedeutet, die Schwierigkeitsstufe wird gesteigert, man gelangt zum nächsten Level.

Ende

Sobald ein Panel nur noch in der obersten Zeile platziert werden kann, ist das Spiel zu Ende. Ziel des Spiels ist es, so viele Panels wie möglich zu platzieren. Mit dem Button PAUSE kann das Spiel unterbrochen werden, eine erneute Betätigung des Buttons lässt das Spiel weiterlaufen.

11.1.2 Programmbeschreibung

Hilfsfeld

Der Kasten, in dem sich die fallenden Panels befinden, ist 8 Spalten breit und 13 Zeilen hoch. Als Hilfskonstruktion steht das zweidimensionale Feld `F` mit 10 Spalten und 15 Zeilen zur Verfügung, in dem jedes existierende Panel mit seiner laufenden Nummer vermerkt ist.

Ze/Sp	0	1	2	3	4	5	6	7	8	9
1	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
2	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
3	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
4	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
5	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
6	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
7	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
8	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
9	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
10	–2	–1	–1	–1	–1	–1	–1	–1	–1	–2
11	–2	–1	–1	–1	11	–1	–1	–1	–1	–2
12	–2	–1	–1	–1	3	8	9	–1	–1	–2
13	–2	–1	0	10	2	4	5	–1	–1	–2
14	–2	–2	–2	–2	–2	–2	–2	–2	–2	–2

Tabelle 11.1 Spielfeld

Im Beispiel in Tabelle 11.1 wird der Inhalt des Felds `F` nach den Panels 0 bis 11, also nach zwölf gefallenen Panels angezeigt. Die Panels 1, 6 und 7 hatten die gleiche Farbe, standen über- oder nebeneinander und sind deshalb schon verschwunden. Die Randelemente werden zu Spielbeginn mit dem Wert der Konstanten `Rand=-2` besetzt. Alle Elemente des Felds `F`, die kein Panel enthalten, also leer sind, haben den Wert der Konstanten `Leer=-1`.

11.1.3 Steuerelemente

Es gibt zu Beginn des Programms folgende Steuerelemente:

- vier Buttons für links (LI), rechts (RE), Drop (DR) und PAUSE
- drei Panels als Begrenzungslinien des Spielfelds

Timer

- einen Timer, der das aktuelle Panel automatisch weiter fallen lässt (Startwert für das Zeitintervall: 500 ms)

Im Verlauf des Programms werden weitere Steuerelemente vom Typ `Panel` hinzugefügt bzw. wieder entfernt.

11.1.4 Initialisierung des Programms

Sie müssen die Namensräume `System.Collections` (für eine ArrayList) und `System.Drawing` (für Positionsänderungen von Steuerelementen) einbinden.

Zu Beginn des Programms werden die klassenweit gültigen Variablen und Konstanten vereinbart, und die `Form1_Load`-Methode wird durchlaufen:

```
public partial class Form1 : Form
{
    ...
    /* Index des aktuellen Panels */
    int PX;

    /* Gesamtes Spielfeld inkl. Randfeldern */
    int[,] F = new int[15, 10];

    /* Zeile und Spalte des aktuellen Panels */
    int PZ, PS;

    /* Schwierigkeitsstufe */
    int Stufe;

    /* Eine zunächst leere Liste von Spiel-Panels */
    ArrayList PL = new ArrayList();

    /* Ein Feld von Farben für die Panels */
    Color[] FarbenFeld = {Color.Red,
        Color.Yellow, Color.Green, Color.Blue,
        Color.Cyan, Color.Magenta, Color.Black,
        Color.White};

    /* Konstanten für Status eines Feldpunkts */
    const int Leer = -1;
    const int Rand = -2;
```



```
    /* Zufallsgenerator erzeugen und initialisieren */
    Random r = new Random();

    private void Form1_Load(...)
    {
        int Z, S;

        /* Größe und Ort einstellen */
        this.Size = new Size(225, 440);
        cmdLinks.Size = new Size(40, 28);
        cmdLinks.Location = new Point(16, 15);
        cmdRechts.Size = new Size(40, 28);
        cmdRechts.Location = new Point(96, 15);
        cmdUnten.Size = new Size(40, 28);
        cmdUnten.Location = new Point(56, 50);
        panLinks.Size = new Size(1, 260);
        panLinks.Location = new Point(20, 80);
        panRechts.Size = new Size(1, 260);
        panRechts.Location = new Point(180, 80);
        panUnten.Size = new Size(160, 1);
        panUnten.Location = new Point(20, 340);
        cmdPause.Size = new Size(70, 28);
        cmdPause.Location = new Point(70, 350);

        /* Feld besetzen */
        for (Z=1; Z<14; Z++)
        {
            F[Z, 0] = Rand;
            for (S=1; S<9; S++)
                F[Z, S] = Leer;
            F[Z, 9] = Rand;
        }

        for (S=0; S<10; S++)
            F[14, S] = Rand;

        /* Initialisierung */
        Stufe = 1;
```

```
            NächstesPanel();
        }
...
```

Listing 11.1 Projekt »Tetris«, Variablen, Konstanten, Start

Zur Erläuterung der klassenweit gültigen Variablen und Konstanten:

- Die laufende Nummer (der Index) des aktuell fallenden Panels wird in der Variablen `PX` festgehalten.

Hilfsfeld

- Das gesamte Spielfeld, das in Abschnitt 11.1.2, »Programmbeschreibung«, schematisch dargestellt wurde, wird im zweidimensionalen Feld `F` gespeichert.
- Die Variablen `PZ` und `PS` beinhalten die Zeilen- und Spaltenposition des aktuell fallenden Panels innerhalb des Spielfelds.

Level

- Die Variable `Stufe` kennzeichnet den Schwierigkeitsgrad des Spiels. Jedes Mal, wenn drei Panels, die untereinander- oder nebeneinanderlagen, gelöscht wurden, wird die Stufe um 1 erhöht. Das sorgt für ein kürzeres Timer-Intervall, die Panels werden schneller.

Liste von Panels

- `PL` ist eine ArrayList von Steuerelementen vom Typ `Panel`. ArrayLists können beliebige Objekte enthalten. Das können Variablen, Objekte eigener Klassen oder, wie hier, Steuerelemente, also Objekte vorhandener Klassen, sein. Zu Beginn ist die ArrayList leer.
- Das Feld `FarbenFeld` enthält insgesamt acht Farben. Die Farben der Panels werden per Zufallsgenerator ermittelt.
- Die Konstanten `Leer` und `Rand` werden erzeugt. Die Namen der Konstanten sind im Programm leichter lesbar als die Werte –1 bzw. –2.
- Für die Farbauswahl wird der Zufallsgenerator bereitgestellt.

Zur Erläuterung der `Form1_Load`-Methode:

- Zunächst werden zur Sicherheit Größe und Ort der beteiligten Steuerelemente noch einmal per Code eingestellt.
- Die Elemente des oben beschriebenen Hilfsfelds `F` werden mit `Leer` bzw. `Rand` besetzt.
- Die Schwierigkeitsstufe wird auf 1 gesetzt.

Erstes Panel

- Es wird die Methode `NächstesPanel()` aufgerufen. Sie ist in diesem Fall für die Erzeugung des ersten fallenden Panels zuständig.

11.1.5 Erzeugen eines neuen Panels

Die Methode `NächstesPanel()` dient der Erzeugung eines neuen fallenden Panels. Das geschieht zu Beginn des Spiels und nachdem ein Panel auf dem unteren Rand des Spielfelds oder auf einem anderen Panel zum Stehen gekommen ist. Der Code lautet:

```
private void NächstesPanel()
{
    int Farbe;
    Panel p = new Panel();

    /* Neues Panel zur ArrayList hinzufügen */
    PL.Add(p);

    /* Neues Panel platzieren */
    p.Location = new Point(100, 80);
    p.Size = new Size(20, 20);

    /* Farbauswahl für neues Panel */
    Farbe = r.Next(0,8);
    p.BackColor = FarbenFeld[Farbe];

    /* Neues Panel zum Formular hinzufügen */
    Controls.Add(p);

    /* Index für späteren Zugriff ermitteln */
    PX = PL.Count - 1;

    /* Aktuelle Zeile, Spalte */
    PZ = 1;
    PS = 5;
}
```

Listing 11.2 Projekt »Tetris«, Methode NächstesPanel

Zur Erläuterung:

- Es wird ein Objekt vom Typ `Panel` neu erzeugt.

Neues Listenelement

- Damit darauf auch außerhalb der Methode zugegriffen werden kann, wird ein Verweis auf dieses Panel mithilfe der Methode `Add()` der ArrayList `PL` hinzugefügt.

- Es werden die Eigenschaften *Ort*, *Größe* und *Farbe* des neuen Panels bestimmt.

Neues Steuerelement

- Das Panel wird mithilfe der Methode `Add()` der Collection `Controls` hinzugefügt. Das ist eine Liste der Steuerelemente des Formulars. Dadurch wird das Panel sichtbar.
- Seine laufende Nummer (der Index) wird mithilfe der Eigenschaft `Count` ermittelt. Diese Nummer wird für den späteren Zugriff benötigt.
- Die Variablen `PZ` und `PS`, die die Position des aktuell fallenden Panels im Spielfeld `F` angeben, werden gesetzt.

11.1.6 Der Zeitgeber

In regelmäßigen Zeitabständen wird das Timer-Ereignis erzeugt und damit die Ereignismethode `timT_Tick()` aufgerufen. Diese sorgt dafür, dass sich das aktuelle Panel nach unten bewegt, falls das noch möglich ist:

```
private void timT_Tick(...)
{
    /* Falls es nicht mehr weitergeht */
    if (F[PZ + 1, PS] != Leer)
    {
        /* Oberste Zeile erreicht */
        if (PZ == 1)
        {
            timT.Enabled = false;
            MessageBox.Show("Das war's");
            return;
        }

        F[PZ, PS] = PX;       // Belegen
        AllePrüfen();
        NächstesPanel();
    }
    else
    {
        /* Falls es noch weitergeht */
        Panel p = (Panel) PL[PX];
        p.Top = p.Top + 20;
```



```
        PZ = PZ + 1;
    }
}
```

Listing 11.3 Projekt »Tetris«, Zeitgeber

Zur Erläuterung:

- Zunächst wird geprüft, ob sich unterhalb des aktuellen Panels noch ein freies Feld befindet.
- Ist das nicht der Fall, hat das Panel seine Endposition erreicht.
- Befindet sich diese Endposition in der obersten Zeile, ist das Spiel zu Ende. Der Timer wird deaktiviert, anderenfalls würden weitere Panels erzeugt. Es erscheint eine Meldung, und die Methode wird unmittelbar beendet. Will der Spieler erneut beginnen, muss er das Programm beenden und neu starten.

Endposition

- Befindet sich die Endposition nicht in der obersten Zeile, wird die Panel-Nummer im Feld `F` mit der aktuellen Zeile und Spalte vermerkt. Das dient der Kennzeichnung eines belegten Feldelements.
- Die Methode `AllePrüfen()` wird aufgerufen (siehe unten), um festzustellen, ob es drei gleichfarbige Panels über- oder nebeneinander gibt. Anschließend wird das nächste Panel erzeugt.

Prüfen

- Befindet sich unterhalb des Panels noch ein freies Feld, kann das Panel weiter fallen. Seine Koordinaten und die aktuelle Zeilennummer werden verändert.

Weiter fallen

11.1.7 Panels löschen

Rekursive Methode

Die Methode `AllePrüfen()` ist eine rekursive Methode, mit deren Hilfe festgestellt wird, ob es drei gleichfarbige Panels nebeneinander oder übereinander gibt. Ist das der Fall, werden diese Panels entfernt, und die darüberliegenden Panels rutschen nach.

Möglicherweise befinden sich nun erneut drei gleichfarbige Panels nebeneinander oder übereinander, es muss also wiederum geprüft werden. Das geschieht so lange, bis keine drei gleichfarbigen Panels nebeneinander oder übereinander mehr gefunden werden.

Die Methode `AllePrüfen()` bedient sich intern der beiden Methoden `NebenPrüfen()` und `ÜberPrüfen()`:

```
private void AllePrüfen()
{
    int Z, S;
    bool Neben, Über;
    Neben = false;
    Über = false;

    /* Drei gleiche Panels nebeneinander ? */
    for(Z=13; Z>0; Z--)
    {
        for(S=1; S<7; S++)
        {
            Neben = NebenPrüfen(Z, S);
            if (Neben) break;
        }
        if (Neben) break;
    }

    /* Drei gleiche Panels übereinander ? */
    for(Z=13; Z>2; Z--)
    {
        for(S=1; S<9; S++)
        {
            Über = ÜberPrüfen(Z, S);
            if (Über) break;
        }
        if (Über) break;
    }

    if (Neben || Über)
    {
        /* Schneller */
        Stufe = Stufe + 1;
        timT.Interval = 5000 / (Stufe + 9);

        /* Eventuell kann jetzt noch eine Reihe
           entfernt werden */
        AllePrüfen();
    }
}
```



```
/* Falls 3 Felder nebeneinander besetzt */
private bool NebenPrüfen(int Z, int S)
{
    int ZX, SX;
    bool ergebnis = false;

    if (F[Z, S] != Leer &&
            F[Z, S + 1] != Leer &&
            F[Z, S + 2] != Leer)
    {
        Panel p = (Panel) PL[F[Z, S]];
        Panel p1 = (Panel) PL[F[Z, S + 1]];
        Panel p2 = (Panel) PL[F[Z, S + 2]];

        /* Falls drei Farben gleich */
        if (p.BackColor == p1.BackColor &&
                p.BackColor == p2.BackColor)
        {

            for(SX=S; SX<S+3; SX++)
            {
                /* PL aus dem Formular löschen */
                Control c = (Control) PL[F[Z, SX]];
                Controls.Remove(c);
                /* Feld leeren */
                F[Z, SX] = Leer;

                /* Panels oberhalb des entladenen
                   Panels absenken */
                ZX = Z - 1;
                while (F[ZX, SX] != Leer)
                {
                    Panel px =
                        (Panel) PL[F[ZX, SX]];
                    px.Top = px.Top + 20;

                    /* Feld neu besetzen */
                    F[ZX + 1, SX] = F[ZX, SX];
                    F[ZX, SX] = Leer;
                    ZX = ZX - 1;
```

```
                }

            }
            ergebnis = true;
        }
    }
    return ergebnis;
}

/* Falls drei Felder übereinander besetzt */
private bool ÜberPrüfen(int Z, int S)
{
    int ZX;
    bool ergebnis = false;

    if (F[Z, S] != Leer && F[Z - 1, S] != Leer &&
            F[Z - 2, S] != Leer)
    {
        Panel p = (Panel) PL[F[Z, S]];
        Panel p1 = (Panel) PL[F[Z - 1, S]];
        Panel p2 = (Panel) PL[F[Z - 2, S]];

        /* Falls drei Farben gleich */
        if (p.BackColor == p1.BackColor &&
                p.BackColor == p2.BackColor)
        {

            /* 3 Panels entladen */
            for (ZX=Z; ZX>Z-3; ZX--)
            {
                /* PL aus dem Formular löschen */
                Control c = (Control) PL[F[ZX, S]];
                Controls.Remove(c);
                /* Feld leeren */
                F[ZX, S] = Leer;
            }
            ergebnis = true;
        }
```



```
    }
    return ergebnis;
}
```

Listing 11.4 Projekt »Tetris«, Panels löschen

Zur Erläuterung:

- Die Variablen `Neben` und `Über` kennzeichnen die Tatsache, dass drei gleichfarbige Panels neben- oder übereinander gefunden wurden. Sie werden erst mal auf `false` gesetzt.
- **Nebeneinander** Zunächst wird geprüft, ob sich drei gleichfarbige Panels nebeneinander befinden. Das geschieht, indem für jedes einzelne Feldelement in der Methode `NebenPrüfen()` geprüft wird, ob es selbst und seine beiden rechten Nachbarn mit einem Panel belegt sind und ob diese Panels gleichfarbig sind. Die Prüfung beginnt beim Panel unten links und setzt sich bis zum drittletzten Panel derselben Zeile fort. Anschließend werden die Panels in der Zeile darüber geprüft usw.
- **Panels löschen** Sobald eine Reihe gleichfarbiger Panels gefunden wurde, werden alle drei Panels mithilfe der Methode `Remove()` aus der Collection der Steuerelemente des Formulars gelöscht, d. h., sie verschwinden aus dem Formular. Ihre Position im Feld `F` wird mit –1 (= `Leer`) besetzt. Nun müssen noch alle Panels, die sich eventuell oberhalb der drei Panels befinden, um eine Position abgesenkt werden. Die Variable `Neben` wird auf `true` gesetzt. Die doppelte Schleife wird sofort verlassen.
- **Übereinander** Analog wird nun in der Methode `ÜberPrüfen()` geprüft, ob sich drei gleichfarbige Panels übereinander befinden. Ist das der Fall, werden sie aus der Collection der Steuerelemente des Formulars gelöscht. Ihre Positionen im Feld `F` werden mit –1 besetzt. Über den drei Panels können sich keine weiteren Panels befinden, die entfernt werden müssten.
- **Rekursiv** Wurde durch eine der beiden Prüfungen eine Reihe gefunden und entfernt, wird die Schwierigkeitsstufe erhöht und das Timer-Intervall verkürzt. Nun muss geprüft werden, ob sich durch das Nachrutschen von Panels wiederum ein Bild mit drei gleichfarbigen Panels über- oder nebeneinander ergeben hat. Die Methode `AllePrüfen()` ruft sich also so lange selbst auf (rekursive Methode), bis keine Reihe mehr gefunden wird.

11.1.8 Panels seitlich bewegen

Mithilfe der beiden Ereignismethoden `cmdLinks_Click()` und `cmdRechts_Click()` werden die Panels nach links bzw. rechts bewegt, falls das möglich ist:

```
private void cmdLinks_Click(...)
{
    if (F[PZ, PS - 1] == Leer)
    {
        Panel p = (Panel) PL[PX];
        p.Left = p.Left - 20;
        PS = PS - 1;
    }
}
private void cmdRechts_Click(...)
{
    if (F[PZ, PS + 1] == Leer)
    {
        Panel p = (Panel) PL[PX];
        p.Left = p.Left + 20;
        PS = PS + 1;
    }
}
```

Listing 11.5 Projekt »Tetris«, Panels seitlich bewegen

Zur Erläuterung:

Seitlich

- Es wird geprüft, ob sich links bzw. rechts vom aktuellen Panel ein freies Feldelement befindet. Ist das der Fall, wird das Panel nach links bzw. rechts verlegt, und die aktuelle Spaltennummer wird verändert.

11.1.9 Panels nach unten bewegen

Die Ereignismethode `cmdUnten_Click()` dient zur wiederholten Bewegung der Panels nach unten, falls das möglich ist. Diese Bewegung wird so lange durchgeführt, bis das Panel auf die Spielfeldbegrenzung oder auf ein anderes Panel stößt. Der Code lautet:


```
private void cmdUnten_Click(...)
{
    while (F[PZ + 1, PS] == Leer)
    {
        Panel p = (Panel) PL[PX];
        p.Top = p.Top + 20;
        PZ = PZ + 1;
    }
    F[PZ, PS] = PX;       // Belegen
    AllePrüfen();
    NächstesPanel();
}
```

Listing 11.6 Projekt »Tetris«, Panels nach unten bewegen

Zur Erläuterung:

Nach unten

- Es wird geprüft, ob sich unter dem aktuellen Panel ein freies Feldelement befindet. Ist das der Fall, wird das Panel nach unten verlegt, und die aktuelle Zeilennummer wird verändert. Das geschieht so lange, bis das Panel auf ein Hindernis stößt.
- Anschließend wird das betreffende Feldelement belegt. Es wird geprüft, ob nun eine neue Reihe von drei gleichfarbigen Panels existiert, und das nächste Panel wird erzeugt.

11.1.10 Pause

Spiel anhalten

Abhängig vom aktuellen Zustand wird durch Betätigen des Buttons PAUSE in den Zustand *Pause* geschaltet oder wieder zurück:

```
private void cmdPause_Click(...)
{
    timT.Enabled = !timT.Enabled;
}
```

Listing 11.7 Projekt »Tetris«, Pause

Zur Erläuterung:

- Der Zustand des Timers wechselt zwischen `Enabled = true` und `Enabled = false`.

Inhalt

10 Zeichnen mit GDI+ 445

11 Beispielprojekte 461

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

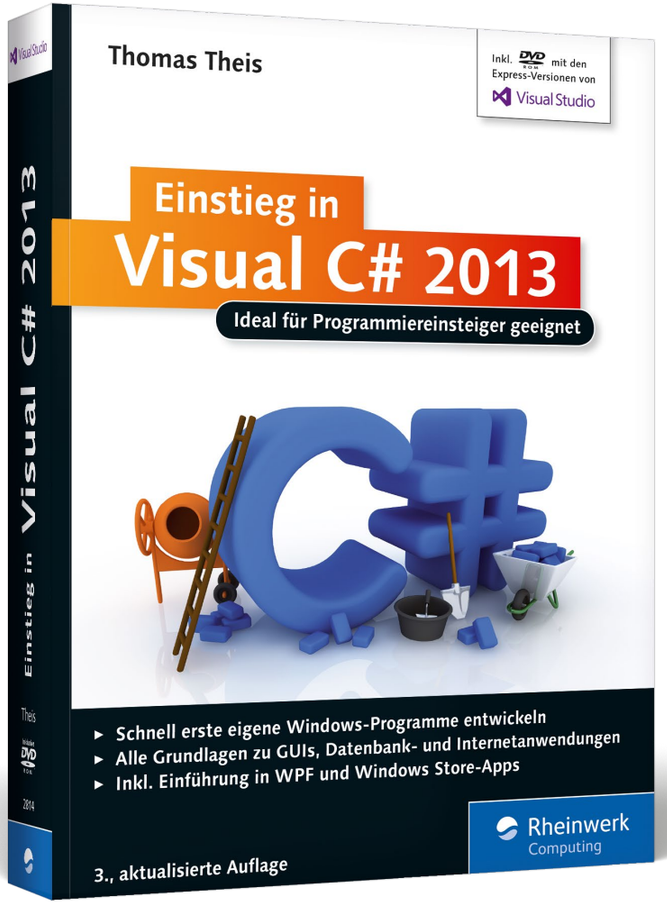
V

W

X

Y

Z



Thomas Theis, Dipl.-Ing. für Technische Informatik, verfügt über langjährige Erfahrung als EDV-Dozent, unter anderem an der Fachhochschule Aachen. Er leitet Schulungen zu C/C++, Visual Basic und zur Webprogrammierung.

Thomas Theis

Einstieg in Visual C# 2013

580 Seiten, broschiert, mit DVD, 3. Auflage 2013
24,90 Euro, ISBN 978-3-8362-2814-5

www.rheinwerk-verlag.de/3573