Thomas Theis

Einstieg in C# mit Visual Studio

Ideal für Programmiereinsteiger

+ Alle Grundlagen zu C# 14 mit GUIs, Objektorientierung und Datenbankanwendungen
+ Programmieren unter .NET mit Visual Studio
+ Mit vielen Beispielprojekten und Übungen

8. Auflage

Beispiele und Musterlösungen zum Download

Rheinwerk Computing

Kapitel 1
Einführung

In diesem Kapitel erlernen Sie anhand eines ersten Projekts den Umgang mit der Entwicklungsumgebung und den Steuerelementen. Anschließend werden Sie in der Lage sein, Ihr erstes eigenes Windows-Programm zu erstellen.

Bei C# (gesprochen: »C Sharp«) handelt es sich um eine objektorientierte Programmiersprache, die im Auftrag von Microsoft entwickelt wurde. In den meisten Fällen wird sie im Zusammenhang mit der .NET-Softwareplattform von Microsoft genutzt.

Die erste Version der Sprache C# erschien im Jahr 2001. Sie hat eine ähnliche Zielsetzung wie Java und C++ und wird ständig weiterentwickelt. In diesem Buch wird mit C# in der Version 14.0 gearbeitet, die zusammen mit der Version 2026 der Entwicklungsumgebung Visual Studio im November 2025 erschienen ist.

In den letzten Jahren wurden der Sprache C# eine Reihe von Eigenschaften und Merkmalen hinzugefügt, besonders im funktionalen Bereich. Damit kann der C#-Code kompakter und verständlicher gestaltet werden.

1.1 C# und Visual Studio

Mithilfe der Entwicklungsumgebung Visual Studio 2026 und der .NET-Softwareplattform können Sie in mehreren Sprachen programmieren, unter anderem in C#. Die .NET-Softwareplattform bietet Klassenbibliotheken, Programmierschnittstellen und Dienstprogramme zur Entwicklung von Anwendungen. Außerdem wird eine Laufzeitumgebung zur Ausführung der Anwendungen zur Verfügung gestellt. In diesem Buch wird mit der .NET-Softwareplattform in der aktuellen Version .NET 10.0 gearbeitet.

Mit C# und der Entwicklungsumgebung Visual Studio 2026 lassen sich Anwendungen unterschiedlichen Typs erstellen, unter anderem:

- *Windows-Forms*-Anwendungen mit leicht zu erstellenden grafischen Benutzeroberflächen und ereignisorientierter Programmierung

- *WPF*-Anwendungen mit der Klassenbibliothek *Windows Presentation Foundation (WPF)* und der Auszeichnungssprache *eXtensible Application Markup Language (XAML)*
- Datenbankanwendungen mit lesendem und schreibendem Zugriff auf unterschiedliche Datenbanksysteme

1.2 Aufbau dieses Buchs

Dieses Buch vermittelt Ihnen zunächst einen einfachen Einstieg in die Programmierung mit C# und der Entwicklungsumgebung Visual Studio 2026. Die Bearbeitung der Beispiele und das selbstständige Lösen der vorliegenden Übungsaufgaben helfen dabei. Dadurch werden Sie schnell erste Erfolgserlebnisse haben, die Sie zum Weitermachen motivieren. In späteren Kapiteln werde ich Ihnen anschließend auch komplexere Themen vermitteln.

Von Anfang an wird mit anschaulichen Windows-Anwendungen gearbeitet. Die Grundlagen der Programmiersprache und die Standardelemente einer Windows-Anwendung, wie Sie sie bereits von anderen Windows-Programmen her kennen, werden gemeinsam vermittelt. Die Anschaulichkeit einer Windows-Anwendung hilft dabei, den eher theoretischen Hintergrund der Programmiersprache leichter zu verstehen.

1.3 Visual Studio 2026

Für dieses Buch wird die frei verfügbare Entwicklungsumgebung *Visual Studio Community 2026* eingesetzt. Sie können sie unter Windows 11 nutzen.

Diese Version von Visual Studio können Sie bei Microsoft herunterladen und auf Ihrem PC installieren. Eine Installationsanleitung finden Sie im Anhang. Die Projekte in diesem Buch wurden unter Windows 11 bearbeitet. Auch die Screenshots sind unter dieser Windows-Version entstanden.

Visual Studio 2026 bietet eine komfortable Entwicklungsumgebung. Sie umfasst einen Editor zur Erstellung des Programmcodes, einen Compiler zur Erstellung der ausführbaren Programme, einen Debugger zur Fehlersuche und vieles mehr. Während der Eingabe Ihres Codes werden Sie im Editor von Hilfsmitteln wie *IntelliSense* und *Copilot* mit vielen wertvollen Hinweisen und Eingabehilfen unterstützt.

Noch eine Anmerkung in eigener Sache: Für die Hilfe bei der Erstellung dieses Buchs bedanke ich mich beim Team des Rheinwerk Verlags, besonders bei Anne Scheibe.

Thomas Theis

1.4 Mein erstes Windows-Programm

Anhand eines ersten Projekts werden Sie nun die verschiedenen Schritte durchlaufen, die zur Erstellung eines einfachen Programms mit C# in Visual Studio notwendig sind. Das Programm soll nach dem Aufruf zunächst so aussehen wie in Abbildung 1.1 gezeigt.

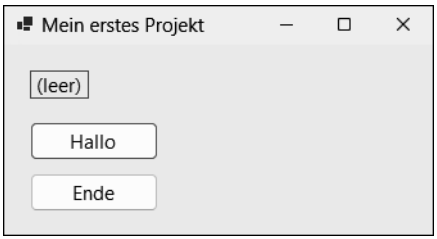

Abbildung 1.1 Erstes Programm nach dem Aufruf

Nach Betätigung des Buttons HALLO soll sich der Text in der obersten Zeile entsprechend verändern (siehe Abbildung 1.2).

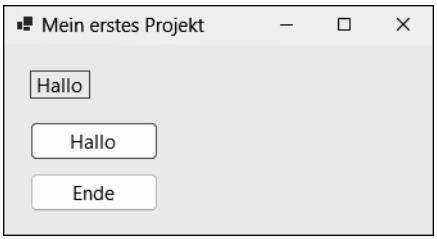

Abbildung 1.2 Nach einem Klick auf den Button »Hallo«

1.5 Visual-Studio-Entwicklungsumgebung

Während der Projekterstellung werden Sie die Visual-Studio-Entwicklungsumgebung Schritt für Schritt kennenlernen.

1.5.1 Ein neues Projekt

Nach dem Aufruf des Programms *Visual Studio Community 2026* betätigen Sie zur Erstellung eines neuen C#-Projekts vom Startbildschirm aus die Schaltfläche NEUES PROJEKT ERSTELLEN (siehe Abbildung 1.3).

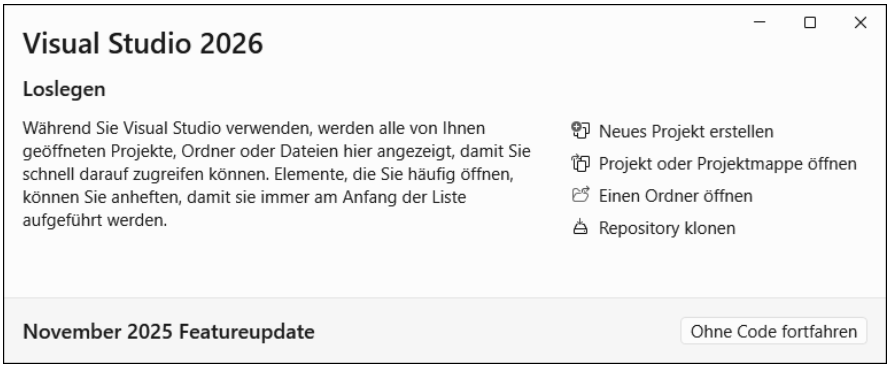

Abbildung 1.3 Startbildschirm

Sollten Sie bereits ein Projekt erstellt und anschließend den Startbildschirm wieder geschlossen haben, steht Ihnen auch der Menüpunkt DATEI • NEU • PROJEKT zur Verfügung.

Anschließend wählen Sie aus der Liste der Vorlagen die Vorlage WINDOWS FORMS-APP aus. Sie ist leichter zu finden, nachdem Sie die Liste der Vorlagen mithilfe der drei Hilfslisten gefiltert haben. Wählen Sie in diesen Hilfslisten wie in Abbildung 1.4 die Einträge C#, WINDOWS und DESKTOP aus.

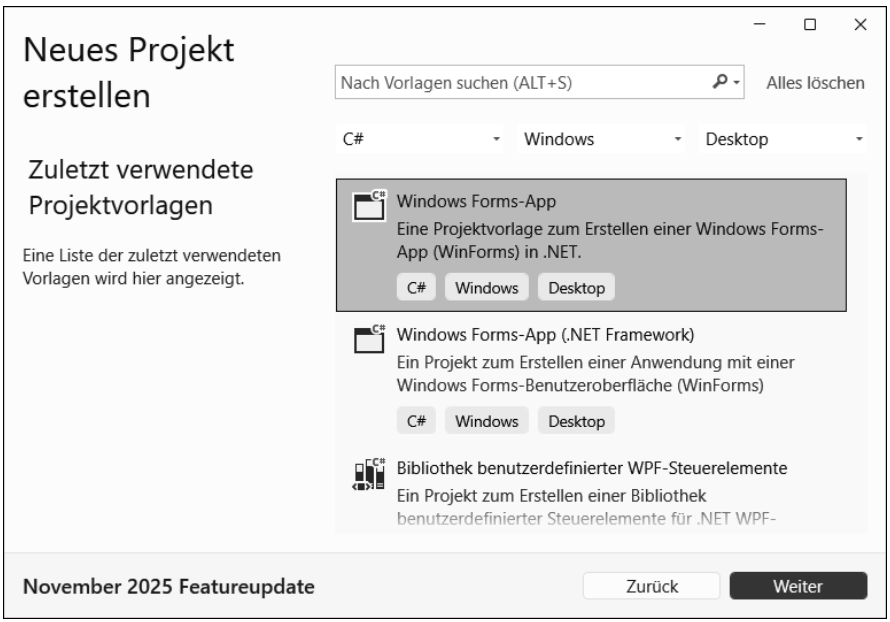

Abbildung 1.4 Vorlage für Windows-Forms-Projekt

Nach Betätigung der Schaltfläche WEITER tragen Sie im nächsten Dialogfeld den Projektnamen ein, hier zum Beispiel: »HalloWelt«, siehe Abbildung 1.5. Zudem wählen Sie den Ort aus, in dem das Verzeichnis für das Projekt neu erstellt wird.

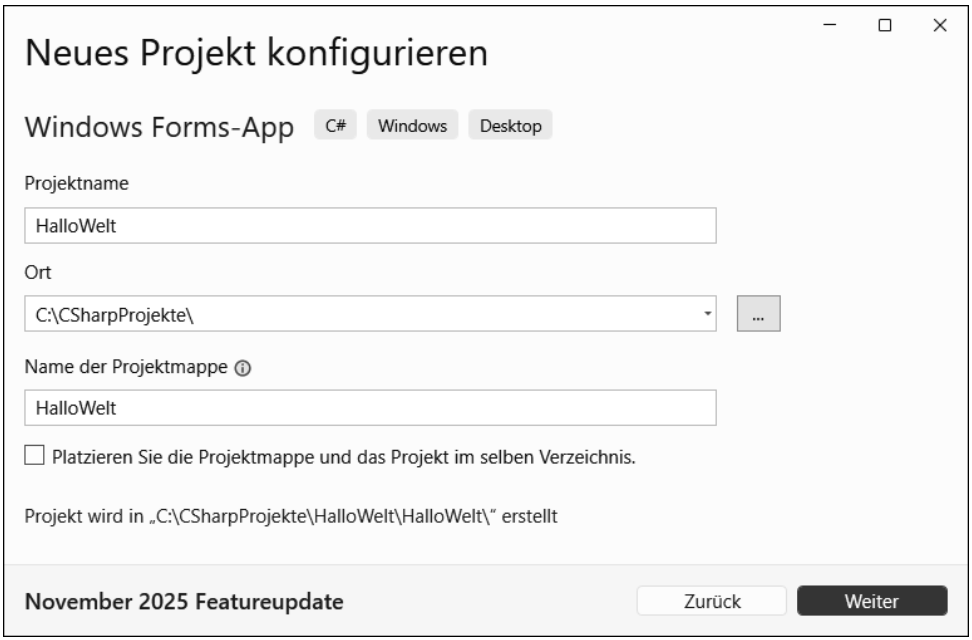

Abbildung 1.5 Projektname und Oberverzeichnis

Wiederum nach der Betätigung der Schaltfläche WEITER wählen Sie das verwendete Framework für Ihr Projekt aus. Nehmen Sie die neueste Version, also .NET 10.0, siehe Abbildung 1.6.

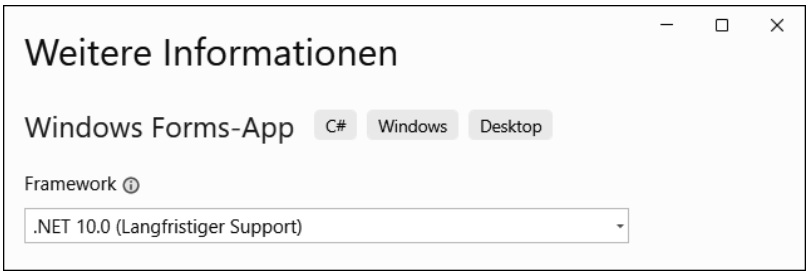

Abbildung 1.6 Auswahl des Zielframeworks

Nach Betätigung der Schaltfläche ERSTELLEN erscheint die Entwicklungsumgebung mit dem neu erstellten Projekt, zunächst mit dem Formular (engl. *form*). Es enthält die Ober-

fläche für die Benutzer des Programms (siehe Abbildung 1.7). Nach dem Erscheinen des Formulars können Sie zwischen der Ansicht des Formulars und der Ansicht des Codes über die Menüpunkte ANSICHT • CODE beziehungsweise ANSICHT • DESIGNER hin- und herschalten.

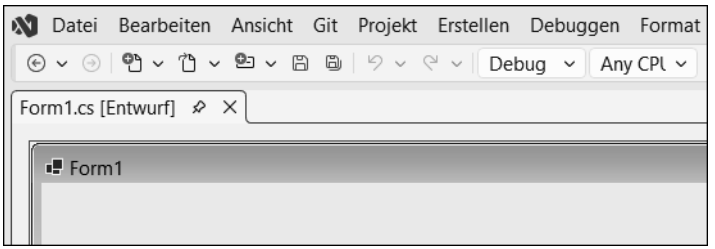

Abbildung 1.7 Benutzerformular

Über das Menü ANSICHT sollten Sie zudem zwei wichtige Hilfsmittel einblenden, und zwar die TOOLBOX und das EIGENSCHAFTEN-Fenster.

Die TOOLBOX (deutsch: Werkzeugkasten) enthält die Steuerelemente für den Benutzer, mit denen er den Ablauf des Programms steuern kann. Sie werden vom Programmentwickler in das Formular eingefügt (siehe Abbildung 1.8).

Sollten in der Toolbox keine Steuerelemente angezeigt werden, klicken Sie einmal auf das Benutzerformular und anschließend wieder auf die Toolbox. Weitere Registerkarten, zum Beispiel DATENQUELLEN, werden nicht benötigt und können jeweils über das Kreuz oben rechts ausgeblendet werden.

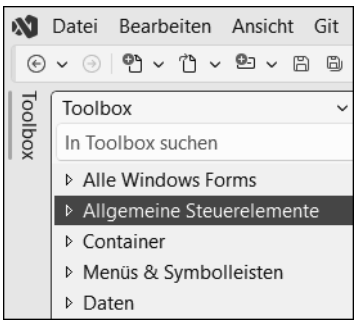

Abbildung 1.8 Verschiedene Kategorien von Steuerelementen

Das EIGENSCHAFTEN-Fenster (engl. *properties window*) dient dem Anzeigen und Ändern der Eigenschaften von Steuerelementen innerhalb des Formulars durch die Programm-

entwicklerin (siehe Abbildung 1.9). Ich empfehle Ihnen, sich die Eigenschaften in alphabetischer Reihenfolge anzeigen zu lassen. Betätigen Sie dazu einfach unter FORM1 SYSTEM.WINDOWS.FORMS.FORM das zweite Symbol von links.

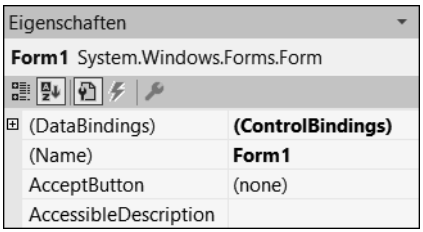

Abbildung 1.9 Eigenschaften-Fenster

Der PROJEKTMAPPEN-EXPLORER (engl. *solution explorer*) zeigt das geöffnete Projekt mit dessen Elementen (siehe Abbildung 1.10). Sollte er nicht angezeigt werden, blenden Sie ihn über das Menü ANSICHT ein.

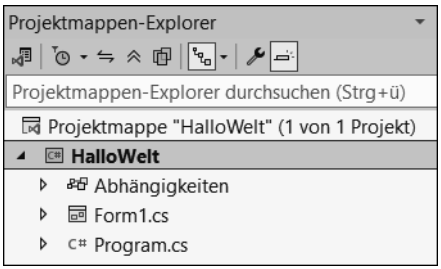

Abbildung 1.10 Projektmappen-Explorer

Ist das Formular nicht sichtbar, führen Sie einen Doppelklick auf den Namen der Formulardatei FORM1.CS im PROJEKTMAPPEN-EXPLORER aus.

Sollten die Eigenschaften eines Steuerelements nicht im bereits sichtbaren EIGENSCHAFTEN-Fenster angezeigt werden, markieren Sie zunächst wiederum den Namen der Formulardatei FORM1.CS im PROJEKTMAPPEN-EXPLORER und anschließend das betreffende Steuerelement. Es empfiehlt sich, den PROJEKTMAPPEN-EXPLORER ein wenig zugunsten des EIGENSCHAFTEN-Fensters zu verkleinern.

1.5.2 Einfügen von Steuerelementen

Zunächst sollen drei Steuerelemente in das Formular eingefügt werden: ein *Bezeichnungsfeld* (*Label*) und zwei *Befehlsschaltflächen* (*Command Buttons*, kurz: *Buttons*). Ein

Bezeichnungsfeld dient dazu, einen Text auf der Benutzeroberfläche anzuzeigen, häufig als Bezeichnung eines anderen Steuerelements. Ein Button dient zum Starten bestimmter Programmteile oder, allgemeiner ausgedrückt, zum Auslösen von Ereignissen.

Um ein Steuerelement einzufügen, ziehen Sie es mithilfe der Maus aus der Toolbox an die gewünschte Stelle im Formular. Alle Steuerelemente finden sich in der Toolbox in der Kategorie Alle Windows Forms. Übersichtlicher ist jedoch der Zugriff über die Kategorie Allgemeine Steuerelemente (engl. *common windows forms*, siehe Abbildung 1.11).

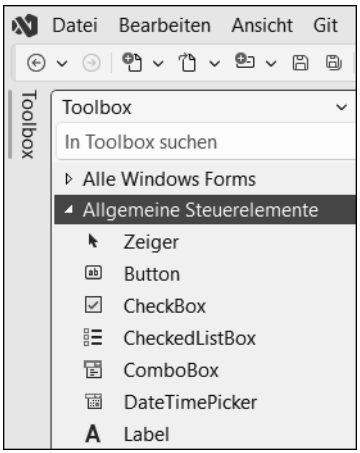

Abbildung 1.11 Allgemeine Steuerelemente

Ein Doppelklick auf ein Steuerelement in der Toolbox fügt es ebenfalls in das Formular ein. Position und Größe des Elements können anschließend noch verändert werden. Dazu wählen Sie das betreffende Steuerelement vorher durch einfaches Anklicken aus (siehe Abbildung 1.12). Ein nicht mehr benötigtes Steuerelement können Sie durch Auswählen und Drücken der Taste `Entf` wieder entfernen.

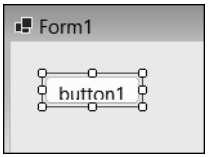

Abbildung 1.12 Ausgewählter Button

Die Größe und andere Eigenschaften des Formulars selbst können Sie ebenfalls verändern. Dazu wählen Sie es vorher durch Anklicken einer freien Stelle im Formular aus.

1.5.3 Arbeiten mit dem Eigenschaften-Fenster

Die eingefügten Steuerelemente haben zunächst einheitliche Namen und Aufschriften; diese sollten Sie für Ihre Programmentwicklung ändern. Es gibt bestimmte Namenskonventionen, die die Lesbarkeit erleichtern: Die Namen enthalten den Typ (mit drei Buchstaben abgekürzt) und die Aufgabe des Steuerelements (jeweils mit großem Anfangsbuchstaben). Ein Button, der die Anzeige der Zeit auslösen soll, wird zum Beispiel mit `CmdZeit` bezeichnet.

Aus den Namen der Steuerelemente ergeben sich auch die Namen der sogenannten *Ereignismethoden*, ebenfalls mit großem Anfangsbuchstaben, siehe Abschnitt 1.5.5. Auf die Einhaltung der Namenskonventionen wird auch im Editor geachtet.

Vorsilben für häufig genutzte Steuerelemente sind:

- `Cmd` für einen Command-Button (deutsch: Befehlsschaltfläche)
- `Txt` für eine TextBox (deutsch: Textfeld)
- `Lbl` für ein Label (deutsch: Bezeichnungsfeld)
- `Opt` für einen RadioButton (deutsch: Optionsschaltfläche)
- `Frm` für ein Form (deutsch: Formular) und
- `Chk` für eine CheckBox (deutsch: Kontrollkästchen)

Zur Änderung des Namens eines Steuerelements muss es zunächst ausgewählt werden. Das können Sie entweder durch einfaches Anklicken des Steuerelements auf dem Formular oder durch seine Auswahl aus der Liste am oberen Ende des EIGENSCHAFTEN-Fensters tun.

Im EIGENSCHAFTEN-Fenster werden alle Eigenschaften des ausgewählten Steuerelements angezeigt. Die Liste ist zweispaltig: In der linken Spalte steht der Name der Eigenschaft, in der rechten ihr aktueller Wert. Die Eigenschaft (NAME) für den Namen des Steuerelements steht am Anfang der Liste der Eigenschaften. Wählen Sie die betreffende Zeile aus, und geben Sie den neuen Namen ein. Nach Bestätigung mit der Taste ↵ ist die Eigenschaft geändert (siehe Abbildung 1.13).

Es ist übersichtlicher, sich die Eigenschaften in alphabetischer Reihenfolge anzeigen zu lassen. Diese Sortierung können Sie wählen, indem Sie das zugehörige Symbol im Kopf des EIGENSCHAFTEN-Fensters anklicken.

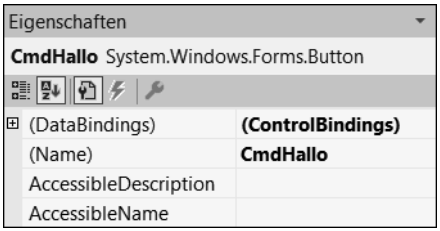

Abbildung 1.13 Button nach der Namensänderung

Die Aufschriften der Buttons, Labels und Formulare entsprechen jeweils den Werten der Eigenschaft `Text`. Die Eigenschaft `Size` (deutsch: Größe) besitzt die Untereigenschaften `Width` (deutsch: Breite) und `Height` (deutsch: Höhe).

Die Eigenschaft `Location` (deutsch: Ort, Position) besitzt die Untereigenschaften `X` (horizontale Position) und `Y` (vertikale Position). Dabei handelt es sich um die Koordinaten der oberen linken Ecke des Steuerelements, gemessen von der oberen linken Ecke des umgebenden Elements (meist des Formulars). Sämtliche Werte werden in Pixeln angegeben.

Wird der Wert einer Eigenschaft geändert, wirkt sich das unmittelbar auf das Steuerelement im Formular aus. Im Folgenden sind die gewünschten Werte für die Eigenschaften der Steuerelemente dieses Programms in Tabellenform angegeben, siehe Tabelle 1.1.

Typ	Eigenschaft	Einstellung
Formular	`Text`	Mein erstes Projekt
	`Size`, `Width` / `Height`	350; 200
Label	`Name`	`LblAnzeige`
	`Text`	(leer)
	`BorderStyle`	`FixedSingle`
	`Location`, `X` / `Y`	20; 20
Button	`Name`	`CmdHallo`
	`Text`	Hallo
	`Size`, `Width` / `Height`	100; 30
	`Location`, `X` / `Y`	20; 60

Tabelle 1.1 Steuerelemente mit Eigenschaften

Typ	Eigenschaft	Einstellung
Button	`Name`	`CmdEnde`
	`Text`	Ende
	`Size`, `Width` / `Height`	100; 30
	`Location`, `X` / `Y`	20; 100

Tabelle 1.1 Steuerelemente mit Eigenschaften (Forts.)

Hiermit legen Sie den Startzustand fest, also diejenigen Werte der Eigenschaften, die die Steuerelemente zu Beginn des Programms beziehungsweise eventuell während des gesamten Programms haben sollen. Viele Eigenschaftswerte können Sie auch noch während der Laufzeit des Programms durch den Programmcode verändern.

Bei einem Label ergibt die Einstellung der Eigenschaft `BorderStyle` auf `FixedSingle` einen Rahmen. Zur Änderung auf `FixedSingle` klappen Sie die Liste bei der Eigenschaft auf und wählen den betreffenden Eintrag aus (siehe Abbildung 1.14). Zur Änderung einiger Eigenschaften müssen Sie gegebenenfalls ein Dialogfeld aufrufen.

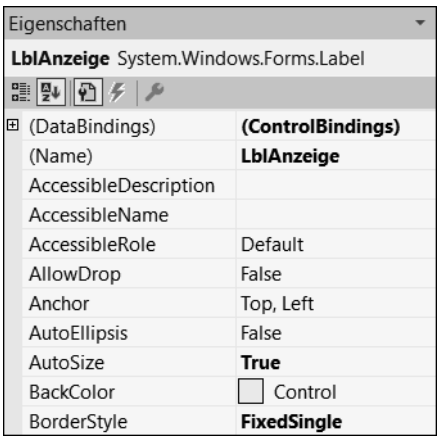

Abbildung 1.14 Label nach der Änderung von Namen und BorderStyle

Im Label soll zunächst der Text (LEER) erscheinen. Hierzu wählen Sie den vorhandenen Text durch Anklicken aus und ändern ihn.

Sie finden alle im aktuellen Formular vorhandenen Steuerelemente in der Liste, die sich am oberen Ende des EIGENSCHAFTEN-Fensters öffnen lässt. Dabei zeigt sich ein Vorteil der einheitlichen Namensvergabe: Die Steuerelemente des gleichen Typs stehen immer direkt untereinander.

1.5.4 Speichern eines Projekts

Die Daten eines C#-Projekts werden innerhalb von Visual Studio in verschiedenen Dateien gespeichert. Zum Speichern des gesamten Projekts verwenden Sie den Menüpunkt DATEI • ALLES SPEICHERN. Diesen Vorgang sollten Sie in regelmäßigen Abständen durchführen, damit keine Änderungen verloren gehen können.

Die in diesem Skript angegebenen Namen erleichtern eine schnelle und eindeutige Orientierung, auch in älteren Programmen.

1.5.5 Das Codefenster

Der Ablauf eines Windows-Programms wird unter anderem durch das Auslösen von Ereignissen durch die Benutzerin gesteuert. Sie löst zum Beispiel die Anzeige des Textes *Hallo* aus, indem sie auf den Button HALLO klickt. Entwickler müssen dafür sorgen, dass aufgrund dieses Ereignisses der gewünschte Text angezeigt wird. Zu diesem Zweck schreiben sie Programmcode und ordnen diesen Code dem Klick-Ereignis zu. Der Code wird in einer sogenannten *Ereignismethode* abgelegt.

Zum Schreiben einer Ereignismethode führen Sie am besten einen Doppelklick auf dem betreffenden Steuerelement aus. Daraufhin erscheint das Codefenster. Zur Erinnerung: Zwischen der Ansicht des Formulars und der Ansicht des Codes können Sie über die Menüpunkte ANSICHT • CODE beziehungsweise ANSICHT • DESIGNER hin- und herschalten. Das ist auch über einen Klick auf den Reiter der betreffenden Registerkarte möglich (siehe Abbildung 1.15).

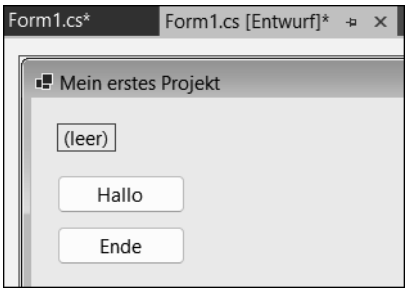

Abbildung 1.15 Registerkarten mit Reitern

Nach erfolgtem Doppelklick auf den Button HALLO erscheinen im Codefenster die folgenden Einträge:

```
namespace HalloWelt
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

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

Listing 1.1 Projekt »HalloWelt«, Button »Hallo«, ohne Code

Lassen Sie sich nicht von der Vielzahl der automatisch erzeugten Zeilen und den noch unbekannten Inhalten abschrecken. Innerhalb der geschweiften Klammern der Ereignismethode `CmdHallo_Click()` wird später Ihr eigener Programmcode hinzugefügt.

Es folgt ein erster Überblick über die anderen Bestandteile, die in späteren Abschnitten für das eigene Programmieren wichtig werden:

- C# ist eine objektorientierte Sprache. Ein wichtiges Element objektorientierter Sprachen sind die sogenannten *Klassen*. Klassen eröffnen weitere Programmiermöglichkeiten. Namensräume (engl. *namespaces*) wiederum enthalten zusammengehörige Klassen.
- Dieses erste Projekt verfügt über einen eigenen Namensraum, daher `namespace HalloWelt`.
- Alle Elemente des aktuellen Formulars `Form1` stehen innerhalb der öffentlich zugänglichen Klasse `Form1`, daher `public class Form1`. Ein Teil der Elemente steht in dieser Datei. Ein anderer Teil, der automatisch erzeugt wurde, steht in einer anderen, hier nicht sichtbaren Datei, daher der Zusatz `partial` (deutsch: teilweise).
- Die Methode `InitializeComponent()` enthält Programmzeilen, die das Aussehen und Verhalten der Steuerelemente des Programms bestimmen.
- Der Zusatz `private` bedeutet, dass die Ereignismethode `CmdHallo_Click()` nur in dieser Klasse bekannt ist. Mit `void` wird gekennzeichnet, dass diese Methode lediglich etwas ausführt, aber kein Ergebnis zurückliefert.

- Auf weitere Einzelheiten dieser automatisch erzeugten Bestandteile werde ich zu einem späteren Zeitpunkt eingehen, da es hier noch nicht notwendig ist und eher verwirren würde.

Der anfänglich ausgeführte Doppelklick führt immer zu dem Ereignis, das am häufigsten mit dem betreffenden Steuerelement verbunden wird. Das ist beim Button das Ereignis `Click`. Zu einem Steuerelement gibt es aber auch noch andere mögliche Ereignisse.

Bei den nachfolgenden Programmen werden nur noch diejenigen Teile im Buch abgebildet, die von der Entwicklerin per Codeeingabe erzeugt werden, sowie diejenigen Teile des automatisch erzeugten Codes, die wichtig für das allgemeine Verständnis sind.

Sie können sich jederzeit den vollständigen Programmcode ansehen, falls Sie eines der Beispielprojekte laden und ausprobieren. Alle Beispielprojekte finden Sie zum Download auf der Webseite zum Buch in den MATERIALIEN.

Hinweis

Lassen sie sich nicht irritieren, falls in Ihrer Installation von Visual Studio blaue Linien unter den deutschen Begriffen im Code erscheinen. Diese Begriffe sind nur für die englische Rechtschreibprüfung unbekannt, beeinflussen aber nicht die korrekte Funktion des Codes oder den Ablauf des Programms. Als mögliche Korrekturmaßnahme wird Ihnen unter anderem eine Deaktivierung der Rechtschreibprüfung vorgeschlagen.

1.5.6 Schreiben von Programmcode

In der Methode `CmdHallo_Click()` soll eine Befehlszeile eingefügt werden, sodass sie anschließend wie folgt aussieht:

```
private void CmdHallo_Click(object sender, EventArgs e)
{
    LblAnzeige.Text = "Hallo";
}
```

Listing 1.2 Projekt »HalloWelt«, Button »Hallo«, mit Code

Der ausgegebene Text muss in Anführungszeichen gesetzt werden.

Der Inhalt einer Methode setzt sich aus einzelnen Anweisungen zusammen, die nacheinander ausgeführt werden. Die vorliegende Methode enthält nur eine Anweisung; in ihr wird mithilfe des Gleichheitszeichens eine Zuweisung durchgeführt.

Bei einer Zuweisung wird der Ausdruck rechts vom Gleichheitszeichen ausgewertet und der Variablen, der Objekteigenschaft oder der Steuerelementeigenschaft links vom Gleichheitszeichen zugewiesen. Die Zeichenkette `Hallo` wird der Eigenschaft `Text` des Steuerelements `LblAnzeige` mittels der Schreibweise `Steuerelement.Eigenschaft = Wert` zugewiesen. Das führt zur Anzeige des Werts.

Nach dem Wechsel auf die Formularansicht können Sie das nächste Steuerelement auswählen, für das eine Ereignismethode geschrieben werden soll. Innerhalb des Codefensters kann Text mit den gängigen Methoden der Textverarbeitung editiert, kopiert, verschoben und gelöscht werden.

In der Ereignismethode `CmdEnde_Click()` soll der folgende Code stehen:

```
private void CmdEnde_Click(object sender, EventArgs e)
{
    Close();
}
```

Listing 1.3 Projekt »HalloWelt«, Button »Ende«

Die Methode `Close()` dient dem Schließen eines Formulars. Da es sich um das einzige Formular dieses Projekts handelt, wird dadurch das Programm beendet und die gesamte Windows-Anwendung geschlossen.

Dies waren einige Beispiele zur Änderung der Eigenschaften eines Steuerelements zur Laufzeit des Programms durch Programmcode. Sie erinnern sich: Zu Beginn hatten wir bereits die Starteigenschaften der Steuerelemente im EIGENSCHAFTEN-Fenster eingestellt.

1.5.7 Kommentare

Bei längeren Programmen mit vielen Anweisungen gehört es zum guten Programmierstil, Kommentarzeilen zu schreiben. In diesen Zeilen werden einzelne Anweisungen oder auch längere Blöcke von Anweisungen erläutert, damit Sie selbst oder auch ein anderer Programmierer sie später leichter nachvollziehen können. Kommentare werden nicht übersetzt oder ausgeführt.

Ein Kommentar beginnt mit der Zeichenkombination /*, endet mit der Zeichenkombination */ und kann sich über mehrere Zeilen erstrecken. Eine andere Möglichkeit ergibt sich durch die Zeichenkombination //. Ein solcher Kommentar erstreckt sich nur bis zum Ende der Zeile.

Der folgende Programmcode wird um einen Kommentar ergänzt:

```
private void CmdEnde_Click(object sender, EventArgs e)
{
    /* Diese Anweisung beendet
       das Programm */
    Close();
}
```

Listing 1.4 Projekt »HalloWelt«, Button »Ende«, mit Kommentar

Hier noch ein kleiner Trick: Sollen bestimmte Programmzeilen für einen Test des Programms kurzfristig nicht ausgeführt werden, können Sie sie auskommentieren, indem Sie die Zeichenkombination // vor die betreffenden Zeilen setzen. Das geht sehr schnell, indem Sie die betreffenden Zeilen markieren und anschließend das entsprechende Symbol in der Symbolleiste anklicken (siehe Abbildung 1.16).

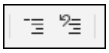

Abbildung 1.16 Kommentar ein/aus

Rechts daneben befindet sich das Symbol, das die Auskommentierung nach dem Test wieder rückgängig macht.

1.5.8 Starten, Ausführen und Beenden des Programms

Nach dem Einfügen der Steuerelemente und dem Erstellen der Ereignismethoden ist das Programm fertig und kann gestartet werden. Dazu betätigen Sie den START-Button in der Symbolleiste (dreieckiger grüner Pfeil nach rechts). Alternativ starten Sie das Programm über die Funktionstaste F5 oder den Menüpunkt DEBUGGEN • DEBUGGEN STARTEN. Das Formular erscheint, und das Betätigen der Buttons führt zum programmierten Ergebnis.

Zur regulären Beendigung eines Programms ist der Button mit der Aufschrift ENDE vorgesehen. Möchten Sie ein Programm während des Verlaufs vorzeitig abbrechen, können Sie auch den ENDE-Button in der Symbolleiste (rotes Quadrat) betätigen. Alternativ beenden Sie das Programm über die Tastenkombination ⇧+F5 oder den Menüpunkt DEBUGGEN • DEBUGGEN BEENDEN.

Tritt während der Ausführung eines Programms ein Fehler auf, werden Sie darauf hingewiesen, und das Codefenster zeigt die entsprechende Ereignismethode sowie die feh-

lerhafte Zeile an. In diesem Fall beenden Sie das Programm, korrigieren den Code und starten das Programm wieder.

Es ist empfehlenswert, das Programm bereits während der Entwicklung mehrmals durch einen Aufruf zu testen und nicht erst, wenn das Programm vollständig erstellt worden ist.

Ein geeigneter Zeitpunkt dazu ergibt sich zum Beispiel

- nach dem Einfügen der Steuerelemente und dem Zuweisen der Eigenschaften, die Sie zu Programmbeginn benötigen, oder
- nach dem Erstellen jeder Ereignismethode.

1.5.9 Ausführbares Programm

Nach erfolgreichem Test des Programms können Sie die ausführbare Datei (*.exe*-Datei) auch außerhalb der Entwicklungsumgebung aufrufen. Haben Sie an den vorgegebenen Einstellungen nichts verändert, findet sie sich im Unterverzeichnis *HalloWelt/bin/Debug/net10.0-windows* des aktuellen Projekts. Das Programm kann mithilfe eines Doppelklicks auf diese Datei im Windows-Explorer gestartet werden.

Die Weitergabe eines eigenen Windows-Programms auf einen anderen PC ist etwas aufwendiger. Diesen Vorgang werde ich im Anhang beschreiben.

1.5.10 Schließen und Öffnen eines Projekts

Um ein Projekt zu schließen, wählen Sie den Menüpunkt Datei • Projektmappe schliessen. Haben Sie Veränderungen vorgenommen, werden Sie vorher gefragt, ob Sie diese Änderungen speichern möchten.

Wollen Sie die Projektdaten sicherheitshalber zwischendurch speichern, ist das über den Menüpunkt Datei • Alles speichern möglich. Das ist bei längeren Entwicklungsphasen sehr zu empfehlen.

Zum Öffnen eines vorhandenen Projekts wählen Sie entweder auf dem Startbildschirm die große Schaltfläche Projekt oder Projektmappe öffnen oder den Menüpunkt Datei • öffnen • Projekt/Projektmappe. Im Dialogfeld Projekt öffnen wählen Sie zunächst das gewünschte Projektverzeichnis aus und anschließend die gleichnamige Projektmappendatei mit der Endung *.slnx*.

Alle Beispielprojekte finden Sie zum Download auf der Webseite zum Buch in den Materialien. Sollte eines der Projekte einmal nicht gestartet werden können, sollten Sie es über den Menüpunkt Erstellen • Projektmappe neu erstellen neu erstellen.

1.5.11 Übung »UName«

Erzeugen Sie ein Windows-Programm mit einem Formular, das zwei Buttons und ein Label enthält (siehe Abbildung 1.17). Bei Betätigung des ersten Buttons erscheint im Label Ihr Name. Bei Betätigung des zweiten Buttons wird das Programm beendet. Hier einige Namensvorschläge: Projektname *UName*, Buttons `CmdMeinName` und `CmdEnde`, Label `LblMeinName`.

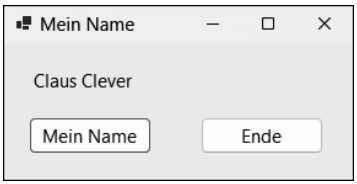

Abbildung 1.17 Übung »UName«

Alle Lösungen zu den Übungsaufgaben finden Sie zum Download auf der Webseite zum Buch in den MATERIALIEN.

1.6 Ausgaben

In C#-Anwendungen werden neben einfachen Texten auch Zahlenwerte, Ergebnisse von Berechnungen, Eigenschaften von Objekten und komplexe Ausdrücke ausgegeben. Im Projekt *GrundlagenAusgaben* in diesem Abschnitt werden einige Regeln für Ausgaben erläutert.

Die Benutzeroberfläche des Programms enthält die vier Buttons `CmdAnzeigen1` bis `CmdAnzeigen4` und ein Label mit dem Namen `LblAnzeige`.

1.6.1 Methode »ToString()«

Nach der Betätigung des ersten Buttons erscheint ein Zahlenwert im Label, siehe Abbildung 1.18. Die Ereignismethode dazu sieht wie folgt aus:

```
private void CmdAnzeigen1_Click(object sender, EventArgs e)
{
    int x = 42;
    // LblAnzeige.Text = x;
    // LblAnzeige.Text = x + "";
    LblAnzeige.Text = x.ToString();
}
```

Listing 1.5 Projekt »GrundlagenAusgaben«, erster Button

In diesem Programm kommt die erste Variable zum Einsatz. Sie hat den Namen `x`. Variablen dienen allgemein zum Speichern von Werten. In Variablen des Datentyps `int` können ganzzahlige Werte gespeichert werden. Mithilfe eines Gleichheitszeichens wird der Variablen `x` der Wert 42 zugewiesen. Mehr zu Variablen und Datentypen folgt in Abschnitt 2.1.

Im Label soll der Wert von `x` ausgegeben werden. Die Zuweisung `LblAnzeige.Text = x` führt zu einem Fehler, da die Eigenschaft `Text` des Labels eine Zeichenkette erwartet und keinen Zahlenwert. Daher ist diese Anweisung auskommentiert.

Die Zahl muss zunächst in eine Zeichenkette umgewandelt werden. Das könnten Sie erreichen, indem Sie `x` mithilfe des Verkettungsoperators `+` mit einer leeren Zeichenkette verbinden. Diese Lösung ist allerdings nicht sehr elegant und daher hier ebenfalls auskommentiert.

Die Methode zur korrekten Umwandlung heißt `ToString()`. Sie wird hier für die Variable `x` aufgerufen und liefert eine Zeichenkette, die der Eigenschaft `Text` des Labels zugewiesen werden kann.

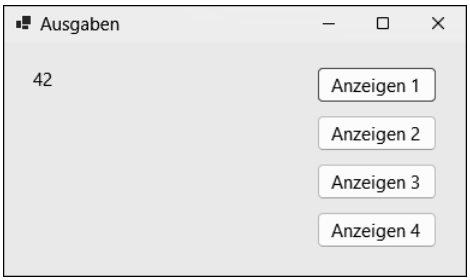

Abbildung 1.18 Ausgabe eines Zahlenwerts

1.6.2 String-Interpolation

Mithilfe der String-Interpolation können Sie Werte von Variablen, Ergebnisse von Berechnungen, Eigenschaften von Objekten und komplexe Ausdrücke unmittelbar und in übersichtlicher Form in Zeichenketten einbetten, siehe Abbildung 1.19. Das geschieht hier nach der Betätigung des zweiten Buttons:

```
private void CmdAnzeigen2_Click(...)
{
    int x = 42;
    LblAnzeige.Text = $"Wert: {x}";
}
```

Listing 1.6 Projekt »GrundlagenAusgaben«, zweiter Button

Die String-Interpolation wird mit dem Operator $ vor der Zeichenkette eingeleitet. Die gewünschten Variablen werden jeweils in geschweifte Klammern gesetzt.

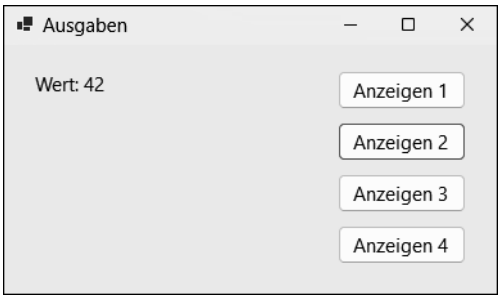

Abbildung 1.19 String-Interpolation

Hinweis

Im Kopfteil vieler Ereignismethoden stehen häufig die Standardparameter `sender` der allgemeinen Klasse `object` und `e` der Klasse `EventArgs`. Sie werden aus Gründen der Übersichtlichkeit im weiteren Verlauf des Buchs durch drei Punkte ersetzt. Sie werden künftig nur noch dargestellt, wenn es auf ihre Inhalte ankommt.

1.6.3 Zeilenumbrüche

Lange Anweisungen im Programmcode können mithilfe von Umbrüchen über mehrere Zeilen verteilt und damit besser lesbar gemacht werden. Das geschieht auch für den Abdruck langer Anweisungen in diesem Buch.

Die Ausgabe eines Programms kann ebenfalls über mehrere Zeilen verteilt werden, und zwar mithilfe der Zeichenfolge `"\n"`, siehe Abbildung 1.20.

Beides wird in der Ereignismethode für den dritten Button gezeigt:

```
private void CmdAnzeigen3_Click(...)
{
    int x = 25, y = 17, z;
    z = x + y;
    LblAnzeige.Text = "Das Ergebnis der " +
        $"Berechnung:\n{x} + {y} = {z}";
}
```

Listing 1.7 Projekt »GrundlagenAusgaben«, dritter Button

Zunächst findet eine Berechnung statt. Die Werte der beiden ganzzahligen Variablen `x` und `y` werden addiert. Das Ergebnis wird in der ganzzahligen Variablen `z` gespeichert. Die gesamte Berechnung wird ausgegeben.

Es handelt sich um eine lange Anweisung, die über zwei Zeilen verteilt wird. Der Umbruch findet in der Zeichenkette statt. Es werden zwei Zeichenketten gebildet, die mithilfe des Verkettungsoperators `+` verbunden werden. Die drei Variablen werden mithilfe der String-Interpolation in der zweiten Zeichenkette eingebettet.

Die Zeichenfolge `"\n"` kann, unabhängig von der String-Interpolation, an beliebigen Stellen in einer Zeichenkette eingebettet werden.

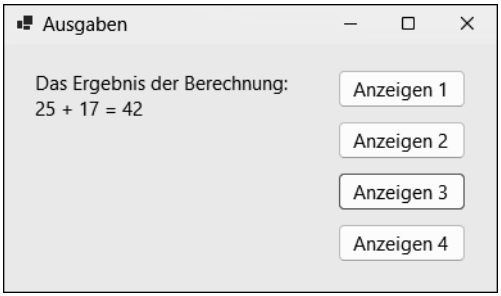

Abbildung 1.20 Zeilenumbrüche

Hinweis

Die Umbrüche zwischen den einzelnen Zeilen einer langen Anweisung können nicht an jeder beliebigen Stelle durchgeführt werden. Ich empfehle die folgenden Stellen:

- nach einer öffnenden Klammer
- vor einer schließenden Klammer
- nach einem Komma in einem Satz
- nach einem Operator
- nach einem Punkt hinter einem Objektnamen

Führen Sie im Editor einen Zeilenumbruch innerhalb einer Zeichenkette durch, werden beide Teile der Zeichenkette automatisch durch Anführungszeichen begrenzt und durch den Verkettungsoperator `+` miteinander verbunden.

1.6.4 Dialogfeld für Ausgabe

Die statische Methode `Show()` der Klasse `MessageBox` bietet die Möglichkeit, eine Ausgabe in einem eigenen Dialogfeld hervorzuheben, siehe Abbildung 1.21. Der Benutzer muss

das Lesen der Information zunächst bestätigen, bevor das Programm weiterläuft. Die Methode erwartet als Parameter innerhalb der runden Klammern eine Zeichenkette. Diese kann nach den beschriebenen Regeln erstellt werden.

Es folgt die Ereignismethode für den vierten Button:

```
private void CmdAnzeigen4_Click(...)
{
    int x = 25, y = 17, z;
    z = x + y;
    MessageBox.Show("Das Ergebnis der " +
        $"Berechnung:\n{x} + {y} = {z}");
    LblAnzeige.Text = "Ende";
}
```

Listing 1.8 Projekt »GrundlagenAusgaben«, vierter Button

Das Dialogfeld kann von Entwicklern auch dazu genutzt werden, während der Entwicklung eines Programms die Werte bestimmter Variablen zu bestimmten Zeitpunkten zu kontrollieren.

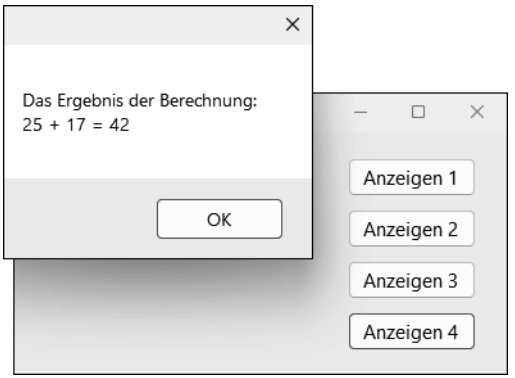

Abbildung 1.21 Dialogfeld für Ausgabe

Hinweis

Mehr zu statischen Elementen einer Klasse finden Sie in Abschnitt 5.9.

1.7 Arbeiten mit Steuerelementen

In diesem Abschnitt lernen Sie anhand eines neuen Projekts mit dem Namen *GrundlagenSteuerelemente* mehr über den Umgang mit Steuerelementen.

1.7.1 Steuerelemente formatieren

Zur besseren Anordnung der Steuerelemente auf dem Formular können Sie sie mithilfe der Maus nach Augenmaß verschieben. Steht dabei das aktuelle Element horizontal oder vertikal parallel zu einem anderen Element, erscheinen automatisch Hilfslinien.

Weitere Möglichkeiten bieten die Menüpunkte im Menü FORMAT. Sollte das Menü nicht sichtbar sein: Markieren Sie in der Formular-Ansicht das Formular oder eines der Steuerelemente, und achten Sie darauf, dass im Eigenschaftenfenster die Eigenschaften des betreffenden Elements angezeigt werden. Spätestens dann wird das Menü FORMAT eingeblendet.

Sollen mehrere Steuerelemente auf einmal formatiert werden, müssen sie zuvor markiert werden (siehe Abbildung 1.22). Das geschieht entweder:

- durch Umrahmung der Elemente mit einem Rechteck, nachdem Sie zuvor das Steuerelement `Zeiger` ausgewählt haben, oder
- durch Mehrfachauswahl, indem Sie ab dem zweiten auszuwählenden Steuerelement die ⇧-Taste (wie für Großbuchstaben) oder die `Strg`-Taste gedrückt halten.

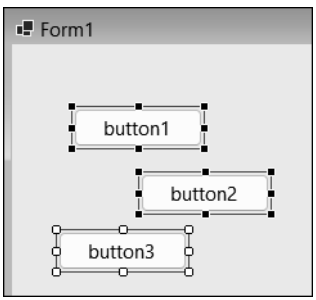

Abbildung 1.22 Mehrere markierte Elemente

Über das Menü FORMAT haben Sie anschließend folgende Möglichkeiten zur Anpassung der Steuerelemente:

- Die ausgewählten Steuerelemente können horizontal oder vertikal zueinander ausgerichtet werden (Menü FORMAT • AUSRICHTEN).

- Auch die horizontalen und/oder vertikalen Dimensionen der ausgewählten Steuerelemente können angeglichen werden (Menü FORMAT • GRÖSSE ANGLEICHEN).
- Zudem können die horizontalen und vertikalen Abstände zwischen den ausgewählten Steuerelementen angeglichen, vergrößert, verkleinert oder entfernt werden (Menü FORMAT • HORIZONTALER ABSTAND/VERTIKALER ABSTAND).
- Die Steuerelemente können horizontal oder vertikal innerhalb des Formulars zentriert werden (Menü FORMAT • AUF FORMULAR ZENTRIEREN).
- Sollten sich die Steuerelemente teilweise überlappen, können Sie einzelne Steuerelemente in den Vorder- beziehungsweise Hintergrund schieben (Menü FORMAT • REIHENFOLGE).
- Sie können alle Steuerelemente gleichzeitig gegen versehentliches Verschieben absichern (Menüpunkt FORMAT • STEUERELEMENTE SPERREN). Diese Sperrung gilt nur während der Entwicklung des Programms.

Abbildung 1.23 zeigt ein Formular mit drei Buttons, die alle linksbündig ausgerichtet sind und den gleichen vertikalen Abstand voneinander haben.

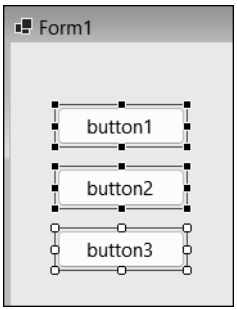

Abbildung 1.23 Nach der Formatierung

Übung

Laden Sie das Projekt *HalloWelt* aus Abschnitt 1.4, markieren Sie darin mehrere Steuerelemente, und testen Sie anschließend die einzelnen Möglichkeiten des FORMAT-Menüs aus.

1.7.2 Steuerelemente kopieren

Zur effektiveren Bearbeitung können vorhandene Steuerelemente einschließlich aller ihrer Eigenschaften kopiert werden. Markieren Sie hierzu die gewünschten Steuerelemente, und kopieren Sie sie über die beiden Menüpunkte BEARBEITEN • KOPIEREN (Tastenkombination Strg+C) und BEARBEITEN • EINFÜGEN (Tastenkombination

Strg+V). Anschließend sollten Sie die neu erzeugten Steuerelemente direkt umbenennen und an den gewünschten Positionen anordnen.

Übung

Laden Sie das Projekt *HalloWelt* aus Abschnitt 1.4, und kopieren Sie einzelne Steuerelemente. Kontrollieren Sie anschließend die Liste der vorhandenen Steuerelemente im EIGENSCHAFTEN-Fenster auf eine einheitliche Namensgebung.

1.7.3 Eigenschaften zur Laufzeit ändern

Steuerelemente haben unter anderem die Eigenschaften `Size` (mit den Komponenten `Width` und `Height`) und `Location` (mit den Komponenten `X` und `Y`) zur Angabe von Größe und Position.

Alle Eigenschaften können sowohl während der Entwicklungszeit als auch während der Laufzeit eines Projekts verändert werden. Zur Änderung während der Entwicklungszeit können Sie die Eigenschaftswerte wie gewohnt im EIGENSCHAFTEN-Fenster eingeben. Als Beispiel für Änderungen während der Laufzeit soll hingegen das folgende Programm im Projekt *GrundlagenSteuerelemente* dienen (siehe Abbildung 1.24).

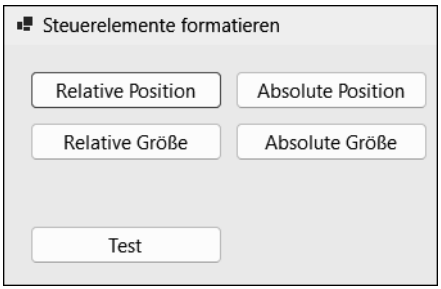

Abbildung 1.24 Position und Größe bestimmen

Es wird nur der Teil des Programmcodes angezeigt, der verändert wurde:

```
private void CmdPositionRel_Click(...)
{
    CmdTest.Location = new Point(
        CmdTest.Location.X + 20, CmdTest.Location.Y);
}
private void CmdPositionAbs_Click(...)
{
    CmdTest.Location = new Point(100, 150);
```

```
}
private void CmdGroesseRel_Click(...)
{
    CmdTest.Size = new Size(
        CmdTest.Size.Width + 20, CmdTest.Size.Height);
}
private void CmdGroesseAbs_Click(...)
{
    CmdTest.Size = new Size(50, 100);
}
```

Listing 1.9 Projekt »GrundlagenSteuerelemente«, Position und Größe

Das Formular enthält zunächst fünf Buttons. Die oberen vier Buttons dienen der Veränderung von Position und Größe des fünften Buttons. Die Position eines Elements kann relativ zur aktuellen Position oder auf absolute Werte eingestellt werden. Das Gleiche gilt für die Größe eines Elements. Bei beiden Angaben handelt es sich um Wertepaare (`X`/`Y` beziehungsweise `Breite`/`Höhe`).

Zur Einstellung der Position dient die Struktur `Point` aus dem Standard-Namensraum `System.Drawing`. Ein Objekt dieser Struktur liefert ein Wertepaar. In diesem Programm wird mit `new` jeweils ein neues Objekt der Struktur `Point` erzeugt, um das Wertepaar bereitzustellen. Zwei Buttons dienen zur Veränderung der Position:

- Bei Betätigung des Buttons Absolute Position wird die Position des fünften Buttons auf die Werte `X=100` und `Y=150` gestellt, jeweils gemessen von der linken oberen Ecke des Formulars.
- Bei Betätigung des Buttons Relative Position wird die Position des fünften Buttons auf die Werte `X = CmdTest.Location.X + 20` und `Y = CmdTest.Location.Y` gestellt. Bei `X` wird also der alte Wert der Komponente `X` um 20 erhöht, das Element bewegt sich nach rechts. Bei `Y` wird der alte Wert der Komponente `Y` nicht verändert, das Element bewegt sich somit nicht nach oben oder unten.

Zur Einstellung der Größe dient die Struktur `Size`, ebenfalls aus dem Namensraum `System.Drawing`. Ein Objekt dieser Struktur liefert ebenfalls ein Wertepaar. Hier wird mit `new` jeweils ein neues Objekt der Struktur `Size` erzeugt, um das Wertepaar bereitzustellen. Zwei Buttons dienen zur Veränderung der Größe:

- Bei Betätigung des Buttons Absolute Grösse wird die Größe des fünften Buttons auf die Werte `Width = 50` und `Height = 100` gestellt.

- Bei Betätigung des Buttons Relative Grösse wird die Größe des fünften Buttons auf die Werte `Width = CmdTest.Size.Width + 20` und `Height = CmdTest.Size.Height` gestellt. Bei `Width` wird also der alte Wert der Komponente `Width` um 20 erhöht, das Element wird breiter. Bei `Height` wird der frühere Wert der Komponente `Height` nicht verändert; das Element verändert seine Höhe daher nicht.

Nach einigen Klicks sieht das Formular aus wie in Abbildung 1.25.

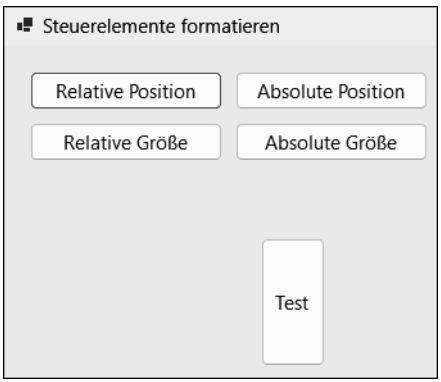

Abbildung 1.25 Veränderung von Eigenschaften zur Laufzeit

1.7.4 Ausgabe von Eigenschaften

In einem Label des Formulars werden die aktuelle Position und Größe des Buttons nach Betätigung des Buttons Anzeigen ausgegeben:

```
private void CmdAnzeigen_Click(...)
{
    LblAnzeige.Text = $"X:{CmdTest.Location.X} " +
        $"Y:{CmdTest.Location.Y}\n" +
        $"Breite:{CmdTest.Size.Width} " +
        $"Höhe:{CmdTest.Size.Height}";
}
```

Listing 1.10 Projekt »GrundlagenSteuerelemente«, Anzeige

Mithilfe des Verkettungsoperators + werden vier Zeichenketten für die Ausgabe miteinander verbunden. Eine einzige lange Zeichenkette hätte ebenfalls ausgereicht, wäre aber nicht so übersichtlich und außerdem für den Abdruck im Buch zu lang gewesen.

Die erste Zeichenkette enthält zunächst den Text "X:". Anschließend folgt dank der String-Interpolation der Wert von CmdTest.Location.X und nicht der Text "CmdTest.Location.X". Die anderen drei Zeichenketten werden auf dieselbe Art gebildet.

Nach einigen Klicks und der Betätigung des Buttons ANZEIGEN sieht das Formular aus wie in Abbildung 1.26.

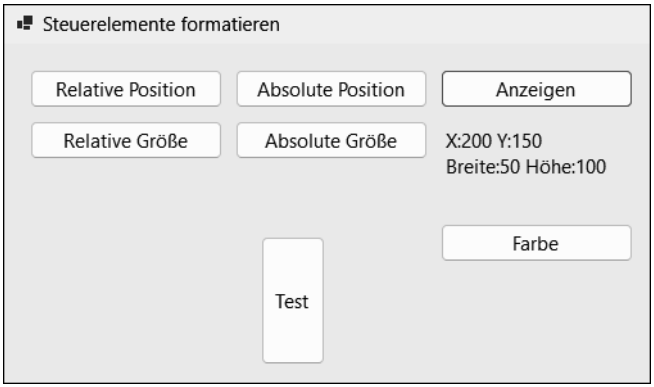

Abbildung 1.26 Anzeige der Eigenschaften

1.7.5 Farben und die Struktur »Color«

Die Hintergrundfarbe eines Steuerelements wird mit der Eigenschaft BackColor festgelegt. Dabei können Sie die Farbe zur Entwicklungszeit wie folgt einstellen: Sie wählen die Eigenschaft BackColor aus, klappen die Liste auf, wählen den Reiter BENUTZERDEFINIERT und danach eine Farbe aus, zum Beispiel Hellrot. Anschließend erscheint der zugehörige Farbcode als Wert der Eigenschaft (hier: 255;128;128). Diese Werte stehen für den Rot-, Grün- und Blau-Anteil der Farbe, jeweils zwischen 0 und 255. Alternativ können Sie für übliche Farben auch den englischen Farbnamen (Red, White, Black ...) eingeben.

Ein Beispiel, ebenfalls im Projekt *GrundlagenSteuerelemente*:

```
private void CmdFarbe_Click(...)
{
    BackColor = Color.Yellow;
    LblAnzeige.BackColor = Color.FromArgb(192, 255, 0);
}
```

Listing 1.11 Projekt »GrundlagenSteuerelemente«, Farbe

Hintergrundfarben und andere Farben können Sie auch zur Laufzeit einstellen. Dabei bedienen Sie sich der Farbwerte aus der Struktur Color, ebenfalls aus dem Namensraum

`System.Drawing`. Sie bietet vordefinierte Farbnamen, zum Beispiel `Yellow`. Der Wert kann der Eigenschaft `BackColor` eines Steuerelements zugewiesen werden.

Die Klasse `Form1` beschreibt das Aussehen und Verhalten des Formulars selbst. Zur Änderung einer Eigenschaft des Formulars müssen Sie nur den Namen der Eigenschaft angeben, ohne den Namen eines Steuerelements davor.

Außerdem bietet die Struktur `Color` die Methode `FromArgb()`. Diese können Sie zum Beispiel mit drei Parametern aufrufen, nämlich den Werten für den Rot-, Grün- und Blau-Anteil der Farbe.

Der betreffende Teil des Formulars sieht nach der Änderung der Farben aus wie in Abbildung 1.27.

Abbildung 1.27 Nach Änderung der Farben

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 während der Laufzeit eines Programms ändern können. Eine Variable besitzt einen eindeutigen Namen, unter dem sie angesprochen werden kann.

2.1.1 Namen und Werte

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 (wie zum Beispiel dem Unterstrich _) bestehen.
- Sie dürfen Umlaute oder auch das »ß« enthalten. Allerdings kann das zu Fehlern beim Einsatz in anderssprachigen Umgebungen führen. Daher rate ich davon ab.
- Innerhalb eines Gültigkeitsbereichs dürfen nicht zwei Variablen mit demselben Namen deklariert werden (siehe Abschnitt 2.1.3).

Variablen erhalten ihre Werte durch Zuweisung per Gleichheitszeichen. Kommt eine Variable auf der rechten Seite des Gleichheitszeichens vor, muss sie vorher einen Wert erhalten haben. Anderenfalls wird ein Fehler gemeldet.

2.1.2 Datentypen

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 zum Beispiel logische Werte speichern möchte. Außerdem muss er sich bei Zahlen Gedanken über die Größe des Zahlenbereichs und die Genauigkeit machen.

Die wichtigsten von C# unterstützten Datentypen können in einige große Gruppen unterteilt werden:

Es gibt Datentypen zur Speicherung von ganzen Zahlen:

- den Datentyp `byte` mit Werten von 0 bis 255
- den Datentyp `short` mit Werten von –32.768 bis 32.767
- den Datentyp `int` mit Werten von –2.147.483.648 bis 2.147.483.647
- den Datentyp `long` mit Werten von –9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807

Außerdem gibt es Datentypen zur Speicherung von Zahlen mit Nachkommastellen:

- den Datentyp `float` mit einfacher Genauigkeit und Werten von ca. $-3{,}4 \times 10^{38}$ bis ca. $3{,}4 \times 10^{38}$
- den Datentyp `double` mit doppelter Genauigkeit und Werten von ca. $-1{,}7 \times 10^{308}$ bis ca. $1{,}7 \times 10^{308}$
- den Datentyp `decimal` mit variabler Genauigkeit und Werten von ca. $-7{,}9 \times 10^{28}$ bis ca. $7{,}9 \times 10^{28}$

Einige weitere nützliche Datentypen sind:

- der Datentyp `bool` für Wahrheitswerte, also `true` oder `false` (wahr oder falsch)
- der Datentyp `char` für einzelne Zeichen
- der Datentyp `string` für Zeichenketten mit variabler Länge

Im folgenden Beispiel werden Variablen dieser Typen deklariert, mit Werten versehen und in einem Label angezeigt (Projekt *GrundlagenDatentypen*).

```
private void CmdAnzeigen_Click(...)
{
    /* Ganze Zahlen */
    byte By;
    short Sh;
```

```
int It, Hex, Bn;
long Lg;

/* Zahlen mit Nachkommastellen */
float Fl;
double Db1, Db2, Exp1, Exp2;
decimal De;

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

/* Ganze Zahlen */
By = 200;
Sh = 30000;
It = 2_000_000_000;
Lg = 3_000_000_000;
Hex = 0x2f5;                // oder: 0x_2f5
Bn = 0b1001;                // oder: 0b_10_01

/* Zahlen mit Nachkommastellen */
Fl = 1.0f / 7;
Db1 = 1 / 7;
Db2 = 1.0 / 7;              // oder: 1.000_000_000
De = 1.0m / 7;
Exp1 = 1.5e3;
Exp2 = 1.5e-3;

/* Boolesche Variable, Zeichen, Zeichenkette */
Bo = true;
Ch = 'a';
St = "Das ist ein Text";

LblAnzeige.Text = $"byte: {By}\n" +
    $"short: {Sh}\n" + $"int: {It}\n" +
    $"long: {Lg}\n" + $"(binäre Zahl): {Bn}\n" +
    $"(hexadezimale Zahl): {Hex}\n\n" +
    $"float: {Fl}\n" + $"double 1: {Db1}\n" +
    $"double 2: {Db2}\n" + $"decimal: {De}\n" +
```

```
        $"Exponent positiv: {Exp1}\n" +
        $"Exponent negativ: {Exp2}\n\n" +
        $"bool: {Bo}\n" + $"char: {Ch}\n" +
        $"string: {St}";
}
```

Listing 2.1 Projekt »GrundlagenDatentypen«

Nach Betätigung des Buttons sieht die Ausgabe wie in Abbildung 2.1 aus.

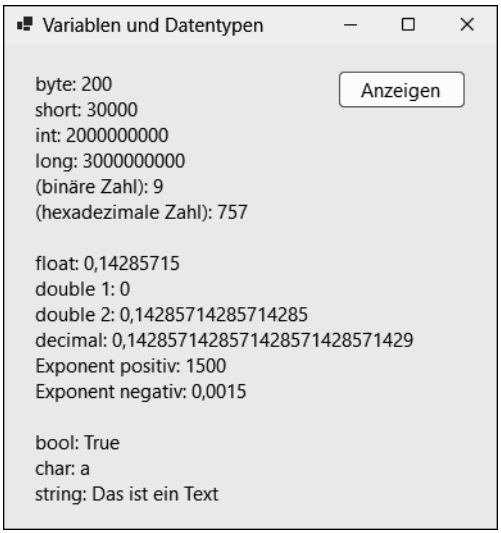

Abbildung 2.1 Wichtige Datentypen

Variablen werden mithilfe der Anweisung `<Datentyp> <Variablenname>;` deklariert. Mehrere Variablen desselben Datentyps können, durch Kommata getrennt, innerhalb einer Anweisung deklariert werden (zum Beispiel `int x, y;`). Variablen können bereits bei der Deklaration mit einem Wert initialisiert werden, zum Beispiel: `short Sh = 30000;`.

Zur deutlicheren Darstellung von Zahlen, die viele Ziffern enthalten, können Sie sowohl vor als auch nach dem Komma das Trennzeichen _ (Unterstrich) nutzen, zum Beispiel als Tausender-Trennzeichen.

Einige Informationen zu ganzen Zahlen:

- Bei den zugehörigen Datentypen führt die Zuweisung einer zu großen Zahl zu einer Überschreitung des Wertebereichs und zu einer Fehlermeldung.
- Ganze Zahlen können auch in hexadezimaler Form zugewiesen werden, mithilfe von 0x zu Beginn der Zahl, gefolgt von den hexadezimalen Ziffern. Diese gehen von 0 bis

9, es folgen a (= dezimal 10), b (= 11), c (= 12), d (= 13), e (= 14) und f (= 15). Ein Beispiel: Die Hexadezimalzahl 0x2f5 entspricht der Dezimalzahl 757, denn es gilt: $2 \times 16^2 + 15 \times 16^1 + 5 \times 16^0 = 512 + 240 + 5 = 757$.

- Eine weitere Möglichkeit zur Zuweisung bieten die binären Zahlen. Sie beginnen mit der Zeichenfolge 0b, gefolgt von binären Ziffern, also 0 oder 1. Ein Beispiel: Die Binärzahl 0b1001 entspricht der Dezimalzahl 9, denn es gilt: $1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 1 = 9$.
- Das Trennzeichen zur deutlicheren Darstellung können Sie auch bei Hexadezimalzahlen oder bei Binärzahlen einsetzen.

Es folgen Informationen über Zahlen mit Nachkommastellen:

- Die zugehörigen Datentypen unterscheiden sich in ihrer Genauigkeit. Nachkommastellen müssen im Programmcode durch einen Dezimalpunkt abgetrennt werden. In der Ausgabe wird dagegen ein Dezimalkomma dargestellt. Die Zuweisung einer zu großen Zahl führt zu einer Fehlermeldung. Die Zuweisung einer zu kleinen Zahl wiederum führt zur Anzeige von Unendlich (!) beziehungsweise zu einer ungenauen Speicherung.
- `float`-Werte sollten mit einem `f` gekennzeichnet werden, `decimal`-Werte mit einem `m`. Damit erhält die gesamte Division im vorliegenden Programm einen `float`- beziehungsweise `decimal`-Wert.
- Sehr große oder sehr kleine Zahlen können im Programmcode auch in der Exponentialschreibweise zugewiesen werden. Zwei Beispiele: `1.5e3` für `1500.0` oder `1.5e-3` für `0.0015`.
- Zahlen mit Nachkommastellen werden nur bis zu einer bestimmten Genauigkeit gespeichert. Daher kann es vorkommen, dass zum Beispiel der berechnete Wert 2,8 als 2,799999999 ausgegeben wird. Sie haben aber die Möglichkeit, Zahlen für die Ausgabe zu formatieren (siehe Abschnitt 4.7.5) beziehungsweise zu runden (siehe Abschnitt 6.6).

Bei der Division von zwei ganzen Zahlen werden die Nachkommastellen abgeschnitten. Möchten Sie das nicht, müssen Sie zumindest eine der beiden Zahlen als Zahl mit Nachkommastellen kennzeichnen, zum Beispiel durch das Anhängen von `.0`: Statt `1` schreiben Sie `1.0`.

Werte für den Datentyp `bool` werden mit `true` und `false` zugewiesen, aber mit `True` und `False` ausgegeben. Werte für einzelne Zeichen müssen in einfachen Anführungszeichen und für Zeichenketten in doppelten Anführungszeichen angegeben werden. Von den hier genannten Datentypen kommen `int`, `double`, `bool` und `string` am häufigsten zum Einsatz.

Sie können Variablen auch mit dem Schlüsselwort `var` deklarieren. Sie müssen dabei mit einem Wert initialisiert werden. Ihr Datentyp ergibt sich implizit mithilfe dieses Werts. In vielen Fällen ergibt sich daraus allerdings zusätzlicher Aufwand bei der Übersetzung oder Ausführung des Programms. Beispiele sehen Sie in Abschnitt 4.8.1.

Übung »UDatentypen«

Schreiben Sie ein Programm im Projekt *UDatentypen*, in dem Ihre Adresse, Ihr Nach- und Vorname, Alter und Gehalt jeweils in Variablen eines geeigneten Datentyps gespeichert und anschließend wie in Abbildung 2.2 ausgegeben werden.

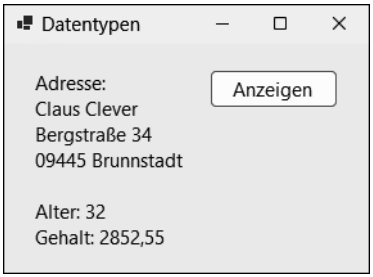

Abbildung 2.2 Übung »UDatentypen«

2.1.3 Gültigkeitsbereich

Eine Variable, die innerhalb einer Methode vereinbart wird, ist nur in ebendieser Methode bekannt und gültig. Außerhalb der Methode ist die Variable unbekannt und ungültig. Eine solche Variable bezeichnet man auch als *lokale Variable*. Sobald die Methode abgearbeitet wurde, steht der Wert der Variablen nicht mehr zur Verfügung. Beim nächsten Aufruf der gleichen Methode wird die Variable neu deklariert und erhält einen neuen Wert.

Eine Variable, die außerhalb einer Methode vereinbart wird, ist innerhalb der gesamten Klasse gültig, hier also innerhalb der Klasse des Formulars. Bei einer solchen Variablen handelt es sich um eine *Eigenschaft des Objekts der Klasse* oder auch *Objekteigenschaft*. Ihr Wert kann in jeder Methode der Klasse gesetzt oder abgerufen werden und bleibt so lange erhalten, wie das Formular im laufenden Programm existiert.

Eine solche Objekteigenschaft kann mit dem Zugriffsmodifizierer `private` deklariert werden. Damit ist sie außerhalb der Klasse unbekannt und ungültig. Wird sie dagegen mit dem Zugriffsmodifizierer `public` vereinbart, ist sie *öffentlich*. Damit ist sie auch außerhalb der jeweiligen Klasse, also zum Beispiel auch in anderen Formularen, bekannt

und gültig. Diese Themen aus dem Bereich der Objektorientierung müssen hier noch nicht weiter vertieft werden, mehr dazu in .

Gibt es in einem Programmabschnitt mehrere Variablen mit demselben Namen, gelten die folgenden Regeln:

- Lokale Variablen mit demselben Namen in derselben Methode sind nicht zulässig.
- Eine Objekteigenschaft wird innerhalb einer Methode von einer lokalen Variablen mit dem gleichen Namen ausgeblendet.

Nachfolgend werden Variablen unterschiedlicher Gültigkeitsbereiche deklariert, verändert und ausgegeben (Projekt *GrundlagenGueltigkeit*):

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

    private void CmdAnzeigen1_Click(...)
    {
        int x = 0;
        Mx++;
        x++;
        LblAnzeige.Text = $"x: {x} Mx: {Mx}";
    }

    private void CmdAnzeigen2_Click(...)
    {
        int Mx = 0;
        Mx++;
        LblAnzeige.Text = $"Mx: {Mx}";
    }
}
```

Listing 2.2 Projekt »GrundlagenGueltigkeit«

In der ersten Methode wird der Wert der Objekteigenschaft `Mx` bei jedem Aufruf erhöht. Die Zuweisung `Mx++` entspricht der Zuweisung `Mx = Mx + 1`, siehe auch Abschnitt 2.2.1. Ebenso entspricht die Zuweisung `x++` der Zuweisung `x = x + 1`, allerdings wird die lokale Variable `x` zu Beginn der Methode immer wieder auf 0 gesetzt. Die Ausgabe des Programms nach mehrfacher Betätigung des ersten Buttons sehen Sie in Abbildung 2.3.

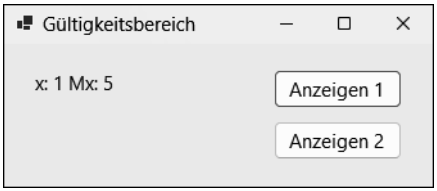

Abbildung 2.3 Lokale Variable »x« und Objekteigenschaft »Mx«

In der zweiten Methode blendet die lokale Variable `Mx` die gleichnamige Objekteigenschaft aus. Die lokale Variable wird zu Beginn der Methode immer wieder auf 0 gesetzt. Die Ausgabe des Programms nach mehrfacher Betätigung des zweiten Buttons sehen Sie in Abbildung 2.4.

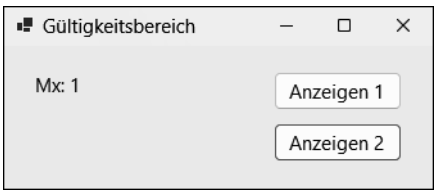

Abbildung 2.4 Lokale Variable »Mx«

Hinweis

Ändern Sie eine Objekteigenschaft im gesamten Formular nicht, macht Visual Studio Sie darauf aufmerksam, dass Sie sie mit dem Attribut `readonly` versehen sollten. Auf diese Weise können Sie sie besser vor einem versehentlichen Schreibzugriff schützen. Hier hätten Sie `Mx` also wie folgt deklariert:

```
private readonly int Mx = 0;
```

Übung »UGueltigkeit«

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

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

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

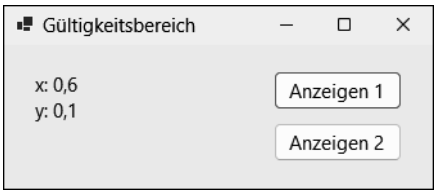

Abbildung 2.5 Ausgabe der ersten Methode nach einigen Klicks

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

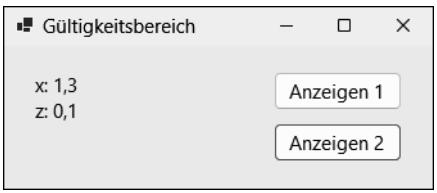

Abbildung 2.6 Ausgabe der zweiten Methode nach weiteren Klicks

2.1.4 Konstanten

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. 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 *GrundlagenKonstanten*):

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

```
    private void CmdAnzeigen1_Click(...)
    {
        const int MaxWert = 55;
        const int MinWert = 5;
        LblAnzeige.Text = $"{(MaxWert - MinWert) / 2}\n{Eintrag}";
    }
}
```

Listing 2.3 Projekt »GrundlagenKonstanten«, Konstanten

Konstanten werden mithilfe des Schlüsselworts `const` definiert.

Die Konstanten `MaxWert` und `Eintrag` werden als konstante Objekteigenschaften deklariert, also als Konstanten mit klassenweiter Gültigkeit. Innerhalb der Methode werden die beiden lokalen Konstanten `MaxWert` und `MinWert` festgelegt. Die lokale Konstante `MaxWert` blendet die konstante Objekteigenschaft gleichen Namens aus, wie Sie in Abbildung 2.7 sehen.

Mithilfe des Operators `$` und der geschweiften Klammern können Sie innerhalb von Zeichenketten auch die Werte von berechneten Ausdrücken oder Umwandlungen angeben.

Visual Studio macht Sie darüber hinaus darauf aufmerksam, dass die konstante Objekteigenschaft `MaxWert` im gesamten Projekt nicht verwendet wird, also entfernt werden könnte.

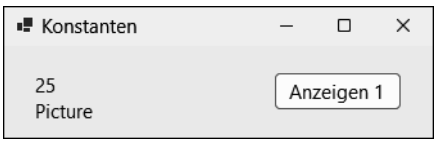

Abbildung 2.7 Konstanten

2.1.5 Enumerationen

Enumerationen sind Aufzählungen von Konstanten, die thematisch zusammengehören. Alle Enumerationen haben denselben Datentyp, der ganzzahlig sein muss. Bei der Deklaration werden ihnen Werte zugewiesen. 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 der Programmiererin, die zahlreichen möglichen Antworten der Benutzerin 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 *GrundlagenKonstanten*):

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

    private void CmdAnzeigen2_Click(...)
    {
        LblAnzeige.Text = $"Farbe: {Farbe.Gelb} {(int)Farbe.Gelb}";
    }

    private void CmdAnzeigen3_Click(...)
    {
        LblAnzeige.Text =
            $"Sonntag: {DayOfWeek.Sunday} {(int)DayOfWeek.Sunday}\n" +
            $"Samstag: {DayOfWeek.Saturday} {(int)DayOfWeek.Saturday}";
    }
}
```

Listing 2.4 Projekt »GrundlagenKonstanten«, Enumerationen

Mithilfe des Schlüsselworts `enum` 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.

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).

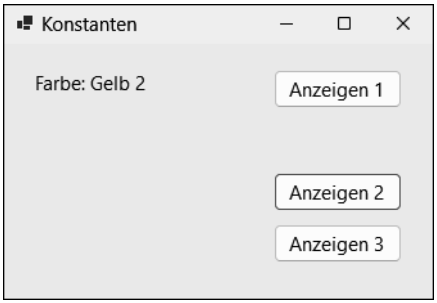

Abbildung 2.8 Erste Enumeration

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.

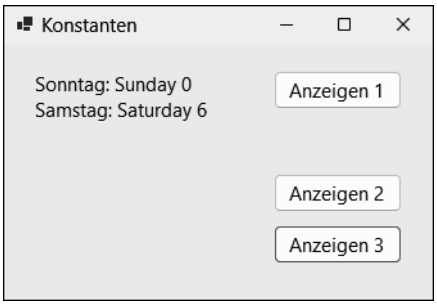

Abbildung 2.9 Zweite Enumeration

2.2 Operatoren

Zum Zusammensetzen von Ausdrücken werden in C# Operatoren aus verschiedenen Kategorien benötigt. Werden mehrere Operatoren innerhalb eines Ausdrucks verwendet, kommen die Vorrangregeln (Prioritäten) für die Reihenfolge der Abarbeitung zum Einsatz, die Sie weiter unten in diesem Abschnitt finden. Sind Sie sich bei der Verwendung dieser Regeln nicht sicher, empfiehlt sich der Einsatz von Klammern zur expliziten Festlegung der Reihenfolge.

2.2.1 Rechenoperatoren

Die Rechenoperatoren aus Tabelle 2.1 werden für Berechnungen eingesetzt.

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

Tabelle 2.1 Rechenoperatoren

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, wenn sie zusammen in einem Ausdruck auftreten. Multiplikation und Division haben Vorrang vor Addition und Subtraktion.

Diese Rangfolge können Sie mit Klammern außer Kraft setzen. Damit erreichen Sie, dass bestimmte Teilausdrücke vor anderen Teilausdrücken ausgewertet werden. In Klammern gesetzte Operationen haben grundsätzlich immer Vorrang. Innerhalb der Klammern gilt jedoch wieder die normale Rangfolge der Operatoren.

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

Der Modulo-Operator `%` berechnet den Rest einer Division. Einige Rechenbeispiele sehen Sie in nachfolgendem Programm:

```
private void CmdAnzeigen1_Click(...)
{
    LblAnzeige.Text =
        $"7 + 2 * 3 = {7 + 2 * 3}\n" +
        $"(7 + 2) * 3 = {(7 + 2) * 3}\n" +
```

```
            $"22 / 8.0 = {22 / 8.0}\n" +
            $"22 / 8 = {22 / 8}\n" +
            $"22 / -8 = {22 / -8}\n" +
            $"19 % 4 = {19 % 4}\n" +
            $"19.5 % 4.5 = {19.5 % 4.5}\n" +
            $"-19.5 % 4.5 = {-19.5 % 4.5}";
}
```

Listing 2.5 Projekt »GrundlagenOperatorenRechnen«

Die Ausgabe dieses Programmteils sehen Sie in Abbildung 2.10.

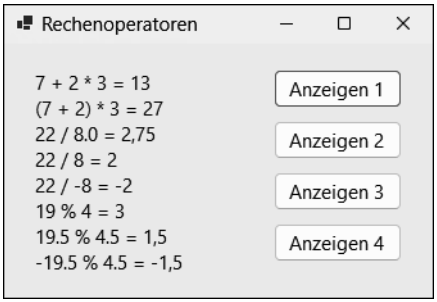

Abbildung 2.10 Wichtige Rechenoperatoren

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

```
private void CmdAnzeigen2_Click(...)
{
    int x = 5;
    x++;
    ++x;
    x = x + 1;
    LblAnzeige.Text = $"Ergebnis: {x}";
}

private void CmdAnzeigen3_Click(...)
{
    int x = 5;
    LblAnzeige.Text = $"Ergebnis: {x++}";
}
```

```
private void CmdAnzeigen4_Click(...)
{
    int x = 5;
    LblAnzeige.Text = $"Ergebnis: {++x}";
}
```

Listing 2.6 Projekt »GrundlagenOperatorenRechnen«

Was passiert in den drei Methoden?

- In der ersten Methode hat `x` zunächst den Wert 5. Der Wert kann mit einer der beiden Kurzformen (`++x` oder `x++`) oder mit `x = x + 1` jeweils um 1 erhöht werden. Anschließend hat `x` den Wert 8. Der Editor schlägt vor, die letzte Anweisung in die Kurzform umzuwandeln.
- 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 noch den alten Wert 5, nach der Anweisungszeile erhält `x` den Wert 6.
- In der dritten Methode wird `x` zunächst um 1 erhöht und anschließend ausgegeben. In diesem Fall steht der Operator `++` vor `x`. In der Ausgabe sehen Sie den neuen Wert 6, nach der Anweisungszeile behält `x` ebenfalls den Wert 6.

Bezüglich der beiden letzten Methoden lässt sich sagen, dass die Schreibweise `x = x + 1;` als eigene Anweisungszeile aufgrund ihrer Eindeutigkeit leichter zu lesen ist. Für den Operator `--` gilt sinngemäß das Gleiche.

Im Projekt *GrundlagenOperatorenRechnen* können Sie auch die beiden Berechnungen mit dem Modulo-Operator `%` selbst nachvollziehen.

Übung »UOperatorenRechnen«

Berechnen Sie im Projekt *UOperatorenRechnen* die beiden Ausdrücke aus Abbildung 2.11, speichern Sie die Ergebnisse in Variablen eines geeigneten Datentyps, und zeigen Sie die Ausdrücke und die Ergebnisse anschließend an.

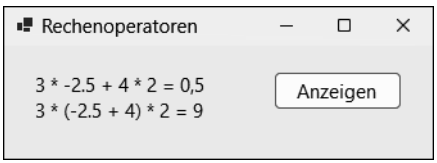

Abbildung 2.11 Rechenoperatoren

2.2.2 Vergleichsoperatoren

Mithilfe von Vergleichsoperatoren (siehe Tabelle 2.2) können Sie feststellen, ob bestimmte Bedingungen zutreffen oder nicht. Das Ergebnis kann beispielsweise zur Ablaufsteuerung von Programmen genutzt werden. In Abschnitt 2.4 werde ich hierauf noch genauer eingehen.

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

Tabelle 2.2 Vergleichsoperatoren

Einige Beispiele sehen Sie in Abbildung 2.12, siehe auch Projekt *GrundlagenOperatorenVergleich*. Anführungszeichen werden in einer Zeichenkette wie folgt gespeichert: `"\"Maier\""`

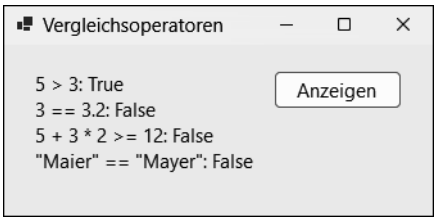

Abbildung 2.12 Vergleichsoperatoren

Übung »UOperatorenVergleich«

Ermitteln Sie im Projekt *UOperatorenVergleich* die Ergebnisse der beiden Ausdrücke aus Abbildung 2.13, speichern Sie sie in Variablen eines geeigneten Datentyps, und zeigen Sie die Ausdrücke und die Ergebnisse anschließend an.

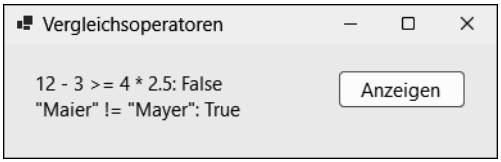

Abbildung 2.13 Vergleichsoperatoren

2.2.3 Logische Operatoren

Logische Operatoren dienen dazu, mehrere Bedingungen zusammenzufassen. Das Ergebnis kann ebenfalls etwa zur Ablaufsteuerung von Programmen genutzt werden (siehe hierzu auch Abschnitt 2.4). Die logischen Operatoren sehen Sie in Tabelle 2.3.

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 2.3 Logische Operatoren

Es seien die Variablen `A = 1`, `B = 3` und `C = 5` des Typs `int` gesetzt. Die Ausdrücke in der ersten Spalte von Tabelle 2.4 ergeben 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 2.4 Ausdrücke mit logischen Operatoren

Aufgrund der höheren Priorität der Vergleichsoperatoren vor den logischen Operatoren `&&`, `||` und `^` werden die Vergleiche zuerst ausgewertet und erst anschließend die Bedingungen zusammengefasst. Der logische Operator `!` hat wiederum eine höhere Priorität als ein Vergleichsoperator. Daher muss der erste Vergleich `A < B` in Klammern gesetzt werden, siehe auch Abschnitt 2.2.5.

In seltenen Fällen werden auch die bitweisen Operatoren & (statt &&) und | (statt ||) eingesetzt. Dabei werden alle Teile eines 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, falls innerhalb des Vergleichsausdrucks Werte verändert werden, zum Beispiel mit den Operatoren ++ oder --, siehe auch Abschnitt 2.2.1.

Alle Berechnungen und Erläuterungen innerhalb dieses Abschnitts können Sie auch mithilfe des Codes im Projekt *GrundlagenOperatorenLogisch* selbst nachvollziehen.

Übung »UOperatorenLogisch«

Ermitteln Sie im Projekt *UOperatorenLogisch* die Ergebnisse der beiden Ausdrücke aus Abbildung 2.14, speichern Sie sie in Variablen eines geeigneten Datentyps, und zeigen Sie die Ausdrücke und die Ergebnisse anschließend an.

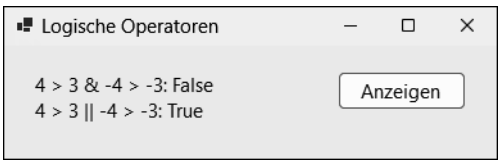

Abbildung 2.14 Logische Operatoren

2.2.4 Zuweisungsoperatoren

Neben dem Zuweisungsoperator = gibt es zur Verkürzung von Anweisungen die kombinierten Zuweisungsoperatoren (siehe Tabelle 2.5), mit denen sogenannte *Verbundzuweisungen* erstellt werden.

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.

Tabelle 2.5 Zuweisungsoperatoren

Operator	Beispiel	Ergebnis
%=	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 2.5 Zuweisungsoperatoren (Forts.)

2.2.5 Rangfolge der Operatoren

Enthält ein Ausdruck mehrere Operatoren, werden die einzelnen Teilausdrücke gemäß der Priorität (= Rangfolge) der Operatoren ausgewertet und aufgelöst, siehe Tabelle 2.6. Je weiter oben die Operatoren in der Tabelle stehen, desto höher ist ihre Priorität.

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

Tabelle 2.6 Priorität der Operatoren

Mit Klammern können Sie diese Rangfolge außer Kraft setzen. Die Ausdrücke innerhalb der Klammern werden als erste ausgewertet.

Übung »UOperatorenRangfolge«

Sind die Bedingungen in Tabelle 2.7 wahr oder falsch? Lösen Sie die Aufgabe möglichst ohne Zuhilfenahme des PC. Sie können Ihre Ergebnisse auch mithilfe des Projekts *UOperatorenRangfolge* kontrollieren.

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 2.7 Übung »UOperatorenRangfolge«

2.3 Einfache Steuerelemente

Die Windows-Programmierung mit C# innerhalb von Visual Studio besteht prinzipiell aus zwei Teilen: der Arbeit mit den visuellen Steuerelementen und der Programmierung mit der Sprache C#. Beides soll in diesem Buch parallel vermittelt werden, um so die eher theoretischen Abschnitte zur Programmiersprache durch anschauliche Praxisbeispiele zu vertiefen.

Daher werde ich zunächst die Steuerelemente Panel, Timer, TextBox und NumericUpDown vorstellen, bevor ich die Verzweigungen zur Programmsteuerung erläutern werde.

2.3.1 Steuerelement »Panel«

Das Steuerelement *Panel* dient normalerweise als Behälter (engl. *container*) für andere Steuerelemente. Sie finden es in der Toolbox in der Kategorie Container. In unserem Beispiel wird es zur visuellen Darstellung eines Rechtecks und für eine kleine Animation genutzt.

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

Mithilfe des nachfolgenden Programms im Projekt *SteuerelementPanel* wird ein Panel durch Betätigung von vier Buttons um zehn Pixel nach oben, unten, links oder rechts

verschoben. Es hat eine Größe von 100 × 100 Pixel sowie eine eigene Hintergrundfarbe und ist in der Mitte des Formulars positioniert. Die Bewegung wird mithilfe der Struktur `Point` durchgeführt.

In Abbildung 2.15 sehen Sie das Panel im Startzustand und in Abbildung 2.16 nach einigen Klicks.

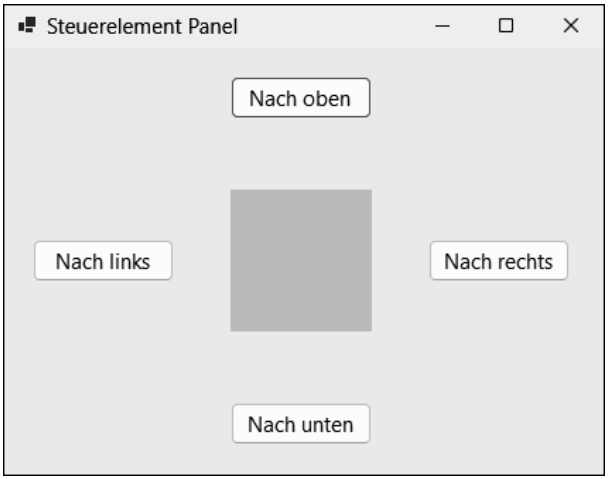

Abbildung 2.15 Zustand zu Beginn

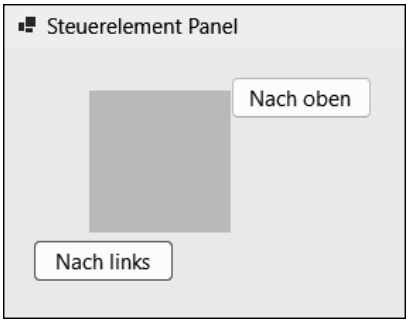

Abbildung 2.16 Zustand nach einigen Klicks

Der Programmcode:

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

Auf einen Blick

Inhalt

2 Grundlagen 49

4 Erweiterte Grundlagen 143

8 Datenbankanwendungen 377

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

Einstieg in C# mit Visual Studio 2026

Programmieren in C# kann ganz einfach sein – das beweist dieses einsteigerfreundliche Buch. Schritt für Schritt lernen Sie an anschaulichen Beispielen, wie Sie C#-Projekte in Visual Studio 2026 erstellen. Schnelle erste Programmiererfolge sind garantiert!

C# und Visual Studio 2026 kennenlernen

Los geht's mit der Installation von Visual Studio 2026. Anschließend lernen Sie alle Grundlagen der C#-Programmierung und schreiben Ihre ersten Programme.

Entwickeln Sie eigene Windows-Programme

Sie lernen die GUI-Entwicklung mit Windows Forms und Windows Presentation Foundation, entwickeln Datenbankanwendungen und erstellen eigene C#-Projekte für .NET 10.

Übung macht den Meister

Ausführliche Schritt-für-Schritt-Anleitungen, anschauliche Beispiele und viele Übungsaufgaben sichern Ihren Lernerfolg. Das Buch ist ideal zum Selbststudium geeignet.

Der Code der Beispiele und die Musterlösungen zu den Übungsaufgaben stehen zum Download bereit.

Thomas Theis ist Dipl.-Ing. für Technische Informatik und arbeitet als Berater und Prüfer im IT-Bereich. Er hat über viele Jahre Kurse für Einsteiger in verschiedenen Programmiersprachen geleitet und ist Autor vieler erfolgreicher Fachbücher.

Erste Schritte

+ Einführung in Visual Studio 2026
+ C#-Sprachgrundlagen
+ Fehlerbehandlung
+ Objektorientiert programmieren
+ Wichtige Klassen in .NET

C#-Programmierung

+ GUIs entwickeln mit Windows Forms
+ Arbeiten mit der kostenfreien Community-Version
+ Datenbankanwendungen
+ Zeichnen mit GDI+

Übungen und Extras

+ Über 130 vollständige Beispielprojekte
+ Übungsaufgaben mit Musterlösungen

Rheinwerk Computing

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

€ 29,90 [D] € 30,80 [A]
Programmierung
ISBN 978-3-367-10849-7