1.5 Mein erstes Windows-Programm

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

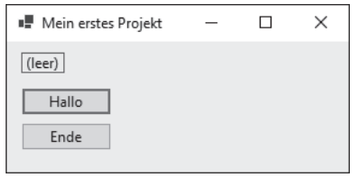

Abbildung 1.1 Erstes Programm nach dem Aufruf

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

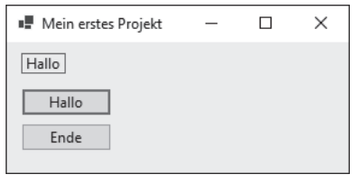

Abbildung 1.2 Nach einem Klick auf den Button »Hallo«

1.6 Visual-Studio-Entwicklungsumgebung

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

1.6.1 Ein neues Projekt

Nach dem Aufruf des Programms *Visual Studio Community 2022* betätigen Sie zur Erstellung eines neuen Visual Basic .NET-Projekts vom Startbildschirm aus die große Schaltfläche NEUES PROJEKT ERSTELLEN (siehe Abbildung 1.3).

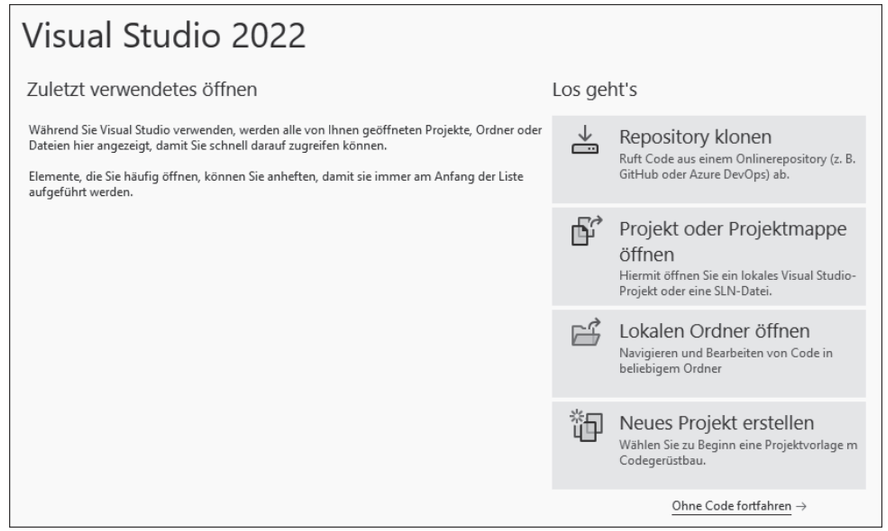

Abbildung 1.3 Startbildschirm

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, siehe Abbildung 1.4. Sie ist leichter zu finden, nachdem Sie die Liste der Vorlagen mithilfe der drei Hilfslisten (am oberen Rand) gefiltert haben. Wählen Sie in diesen Hilfslisten die Einträge VISUAL BASIC, 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 das Oberverzeichnis aus, in dem das Verzeichnis für das Projekt neu erstellt wird.

Abbildung 1.5 Projektname und Oberverzeichnis

Nach erneuter Betätigung der Schaltfläche WEITER wählen Sie die Ziel-Plattform für Ihr Projekt aus. Zur Nutzung der neuesten Version der .NET-Softwareplattform und der Sprache Visual Basic .NET wählen Sie .NET 6.0, siehe Abbildung 1.6.

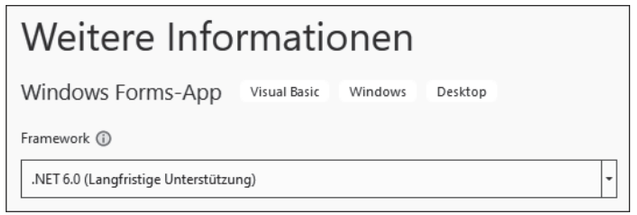

Abbildung 1.6 Ziel-Plattform

Nach einem Klick auf die Schaltfläche ERSTELLEN erscheint nach kurzer Zeit die Entwicklungsumgebung mit dem neu erstellten Projekt und dem zugehörigen Formular (engl. *form*). Das Formular enthält die Oberfläche 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.

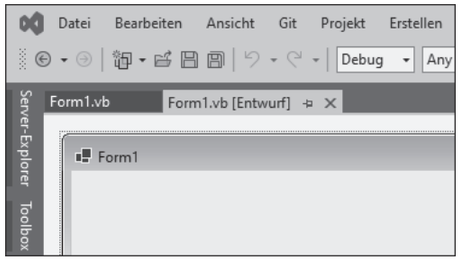

Abbildung 1.7 Benutzerformular

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

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

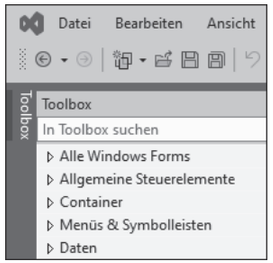

Abbildung 1.8 Verschiedene Kategorien von Steuerelementen

Das EIGENSCHAFTEN-Fenster (engl. *properties window*) dient dem Anzeigen und Ändern der Eigenschaften von Steuerelementen innerhalb des Formulars durch die Programmentwicklerin (siehe Abbildung 1.9). Ich empfehle Ihnen, sich die Eigenschaften in alphabetischer Reihenfolge anzeigen zu lassen. Betätigen Sie dazu einfach unter FORM1 das zweite Symbol von links.

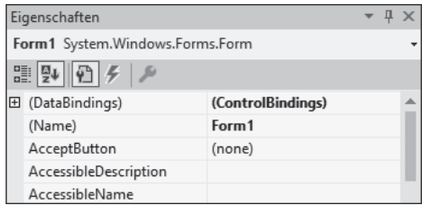

Abbildung 1.9 »Eigenschaften«-Fenster

Der Projektmappen-Explorer (engl. *solution explorer*) zeigt das geöffnete Projekt mit dessen Elementen (siehe Abbildung 1.10).

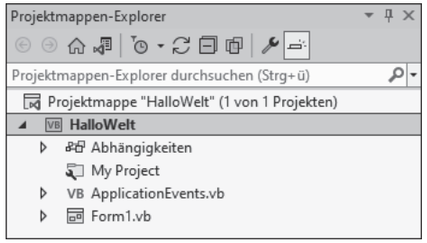

Abbildung 1.10 Projektmappen-Explorer

Sollte die Toolbox, das Eigenschaften-Fenster oder der Projektmappen-Explorer nicht angezeigt werden, können Sie das betreffende Element über das Menü Ansicht einblenden. Ist das Formular nicht sichtbar, blenden Sie es einfach über einen Doppelklick auf den Namen der Formulardatei Form1.vb im Projektmappen-Explorer ein.

Werden die Eigenschaften eines Steuerelements nicht im bereits sichtbaren Eigenschaften-Fenster angezeigt, markieren Sie zunächst wiederum den Namen der Formulardatei Form1.vb im Projektmappen-Explorer und anschließend das betreffende Steuerelement. Es empfiehlt sich, den Projektmappen-Explorer ein wenig zugunsten des Eigenschaften-Fensters zu verkleinern.

1.6.2 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 unter Alle Windows Forms. Übersichtlicher ist jedoch der Zugriff über Allgemeine Steuerelemente (engl. *common windows forms*, siehe Abbildung 1.11).

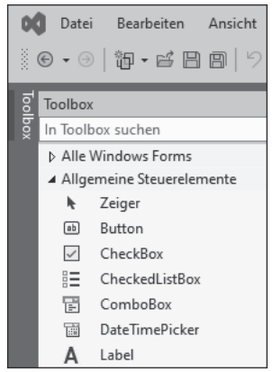

Abbildung 1.11 Allgemeine Steuerelemente

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.

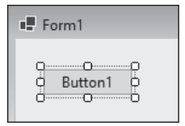

Abbildung 1.12 Ausgewählter Button

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.

1.6.3 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 An-

fangsbuchstaben). 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 1.6.5, »Das Codefenster«. 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)
- `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 Auswahl desselben aus der Liste am oberen Ende des EIGENSCHAFTEN-Fensters erreichen.

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).

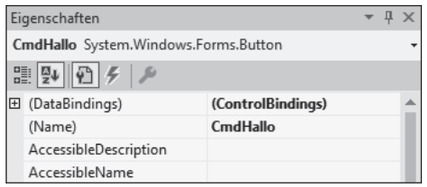

Abbildung 1.13 Button nach der Namensänderung

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). 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`	300
	`Size.Height`	200
Button	`Name`	`CmdHallo`
	`Text`	Hallo
Button	`Name`	`CmdEnde`
	`Text`	Ende
Label	`Name`	`LblAnzeige`
	`Text`	(leer)
	`BorderStyle`	`FixedSingle`

Tabelle 1.1 Steuerelemente mit Eigenschaften

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.

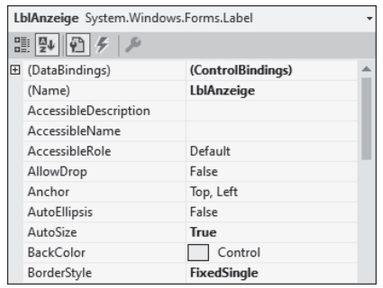

Abbildung 1.14 Label nach der Änderung von Namen und BorderStyle

Im Label soll zunächst der Text (LEER) erscheinen. Hierzu wählen Sie die zugehörige Eigenschaft TEXT aus und ändern ihren Wert.

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.

1.6.4 Speichern eines Projekts

Die Daten eines Visual Basic .NET-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.

1.6.5 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 Texts *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 die Registerkarte oberhalb des Formulars beziehungsweise des Codefensters möglich (siehe Abbildung 1.15).

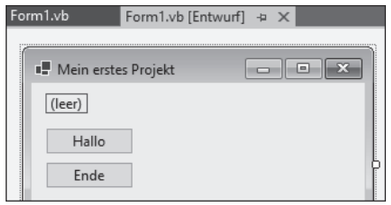

Abbildung 1.15 Registerkarten

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

```
Public Class Form1
    Private Sub CmdHallo_Click(sender As Object,
            e As EventArgs) Handles CmdHallo.Click


    End Sub
End Class
```

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

Lassen Sie sich nicht von der Vielzahl der automatisch erzeugten Zeilen und den noch unbekannten Inhalten abschrecken. Innerhalb der Ereignismethode `CmdHallo_Click()` wird später Ihr eigener Programmcode hinzugefügt. Für den Druck in diesem Buch wurde die lange Zeile mit `Private Sub` auf zwei Zeilen verteilt.

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

- Visual Basic .NET ist eine objektorientierte Sprache. Ein wichtiges Element objektorientierter Sprachen sind die sogenannten *Klassen*. Klassen eröffnen weitere Programmiermöglichkeiten.
- In Visual Basic .NET wird eine Klasse mit `Class` eingeleitet und mit `End Class` beendet. Alle Elemente des aktuellen Formulars `Form1` stehen innerhalb der öffentlich zugänglichen Klasse `Form1`, daher `Public Class Form1`.
- Der Zusatz `Private` bedeutet, dass die Ereignismethode `CmdHallo_Click()` nur in dieser Klasse bekannt ist. In Visual Basic .NET werden die Ereignismethoden mithilfe von Prozeduren gebildet. Eine Prozedur wird mit `Sub` eingeleitet und mit `End Sub` beendet. Eine Prozedur ist eine Methode, die lediglich etwas ausführt, aber kein Ergebnis zurückliefert.
- Das Schlüsselwort `Handles` dient zur Verbindung von Methode und Ereignis. Im vorliegenden Beispiel wird das Ereignis `Click` des Buttons `CmdHallo` mit der Methode `CmdHallo_Click()` verbunden.
- Auf weitere Einzelheiten dieser automatisch erzeugten Bestandteile werde ich zu einem späteren Zeitpunkt eingehen, da dies 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 zu diesem Buch in den MATERIALIEN.

1.6.6 Schreiben von Programmcode

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

```
Private Sub CmdHallo_Click(sender As Object,
        e As EventArgs) Handles CmdHallo.Click
    LblAnzeige.Text = "Hallo"
End Sub
```

Listing 1.2 Projekt »HalloWelt«, Button »Hallo«, mit eigenem 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 Sub CmdEnde_Click(sender As Object,
        e As EventArgs) Handles CmdEnde.Click
    Close()
End Sub
```

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.

1.6.7 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 kann. Kommentare werden nicht übersetzt oder ausgeführt.

Ein Kommentar beginnt mit einem Hochkomma, also dem Zeichen ', und erstreckt sich bis zum Ende der Zeile. Nachfolgend wird der Programmcode um einen Kommentar ergänzt:

```
Private Sub CmdEnde_Click(sender As Object,
        e As EventArgs) Handles CmdEnde.Click
    ' Diese Anweisung beendet das Programm
    Close()
End Sub
```

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 ein Hochkomma 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.

1.6.8 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 (nach rechts weisender dreieckiger grüner Pfeil). 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 hierauf hingewiesen, und das Codefenster zeigt die entsprechende Ereignismethode sowie die fehlerhafte Zeile an. In diesem Fall beenden Sie das Programm, korrigieren den Code und starten das Programm erneut.

Es empfiehlt sich, das Programm bereits während der Entwicklung mehrmals durch einen Aufruf zu testen und nicht erst, wenn es vollständig erstellt wurde.

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.

1.6.9 Ausführbares Programm

Nach dem erfolgreichen 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/net6.0-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.

1.6.10 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. Bei längeren Entwicklungsphasen ist das 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 *.sln*.

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 einfach neu erstellen.

1.6.11 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. Einige Namensvorschläge: Projektname *UName*, Buttons `CmdMeinName` und `CmdEnde`, Label `LblMeinName`.

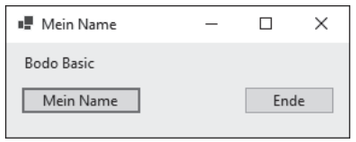

Abbildung 1.17 Übung »UName«

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

1.7 Ausgaben

In Visual Basic .NET-Anwendungen werden neben einfachen Texten auch Zahlenwerte, Ergebnisse von Berechnungen, Eigenschaften von Objekten und komplexe Ausdrücke ausgegeben. In dem 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`.

1.7.1 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 Sub CmdAnzeigen1_Click(sender As Object,
        e As EventArgs) Handles CmdAnzeigen1.Click
    Dim x As Integer
    x = 42
    ' LblAnzeige.Text = x
    ' LblAnzeige.Text = x & ""
    LblAnzeige.Text = x.ToString()
End Sub
```

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. Eine Variable kann mithilfe von `Dim ... As ...` deklariert werden. Bei einer Deklaration erhält eine Variable ihren Namen und ihren Datentyp. In Variablen des Datentyps `Integer` 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` ist möglich, da der ganzzahlige Wert der Variablen automatisch in eine Zeichenkette umgewandelt wird, die der Eigenschaft `Text` des Labels zugewiesen wird.

Der Verkettungsoperator `&` dient standardmäßig zum Verbinden mehrerer Zeichenketten. Wird der Wert einer Zahlenvariablen mit einer (hier leeren) Zeichenkette verbunden, findet ebenfalls eine automatische Umwandlung der Zahl in eine Zeichenkette statt. Die entstandene Zeichenkette wird wiederum der Eigenschaft `Text` des Labels zugewiesen.

Besser ist es, die Zahl explizit mithilfe der Methode `ToString()` in eine Zeichenkette umzuwandeln. Die Methode 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

1.7.2 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 Sub CmdAnzeigen2_Click(...) Handles ...
    Dim x = 42
    LblAnzeige.Text = $"Wert: {x}"
End Sub
```

Listing 1.6 Projekt »GrundlagenAusgaben«, zweiter Button

Wird einer Variablen bereits bei ihrer Deklaration ein Wert zugewiesen, ergibt sich ihr Datentyp implizit aus dem Datentyp des Werts. Die Zahl 42 ist ganzzahlig, daher ist `x` vom Typ `Integer`.

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` des Typs `Object` und `e` des Typs `EventArgs` sowie das Ereignis `Click` eines Buttons. Aus Gründen der Übersichtlichkeit werden diese Elemente im weiteren Verlauf des Buchs jeweils durch drei Punkte ersetzt. Sie werden künftig nur noch dargestellt, wenn es auf diese Elemente ankommt.

1.7.3 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 Konstanten `vbCrLf`, siehe Abbildung 1.20. Eine Konstante ist eine unveränderliche Variable. Sie können eigene Konstanten definieren oder eine der vielen integrierten Konstanten nutzen.

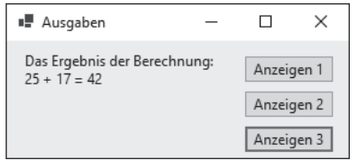

Abbildung 1.20 Zeilenumbrüche

Es folgt die Ereignismethode für den dritten Button:

```
Private Sub CmdAnzeigen3_Click(...) Handles ...
    Dim x = 25, y = 17, z As Integer
    z = x + y
    LblAnzeige.Text = "Das Ergebnis der " &
        $"Berechnung:{vbCrLf}{x} + {y} = {z}"
End Sub
```

Listing 1.7 Projekt »GrundlagenAusgaben«, dritter Button

Sie können mithilfe von `Dim` in einer Anweisung mehrere Variablen deklarieren. Dabei müssen die Variablen durch Kommata getrennt werden. Hier erhalten die Variablen `x` und `y` ihren Datentyp `Integer` durch den zugewiesenen Wert, die Variable `z` mithilfe von `As`.

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

Bei der Ausgabe handelt es 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. Dabei muss `&` am Ende der ersten Zeile stehen, nicht erst am Beginn der zweiten Zeile. Die drei Variablen und die Konstante werden mithilfe der String-Interpolation in der zweiten Zeichenkette eingebettet.

Hinweis

Der Name der Konstanten `vbCrLf` setzt sich aus drei Teilen zusammen. Mit `vb` beginnt der Name von allgemeinen Konstanten in Visual Basic .NET. `Cr` steht für *carriage return* (deutsch: Wagenrücklauf) und `Lf` für `Line feed` (deutsch: Zeilenvorschub).

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

1.7.4 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 Sub CmdAnzeigen4_Click(...) Handles ...
    Dim x = 25, y = 17, z As Integer
    z = x + y
    MessageBox.Show("Das Ergebnis der " &
        $"Berechnung:{vbCrLf}{x} + {y} = {z}")
    LblAnzeige.Text = "Ende"
End Sub
```

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.

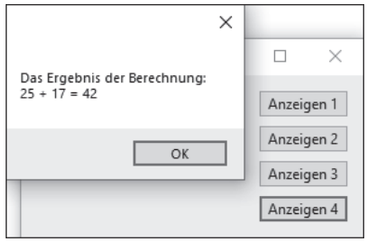

Abbildung 1.21 Dialogfeld für Ausgabe

> **Hinweis**
>
> Mehr zu statischen Elementen einer Klasse finden Sie in Abschnitt 5.8.

1.7.5 Fehler behandeln

Sollten Sie in einem Ihrer Projekte eine Fehlermeldung übersehen haben und dennoch die Erstellung und Ausführung des Projekts starten, erscheint ein Dialogfeld mit einem entsprechenden Hinweis, siehe Abbildung 1.22.

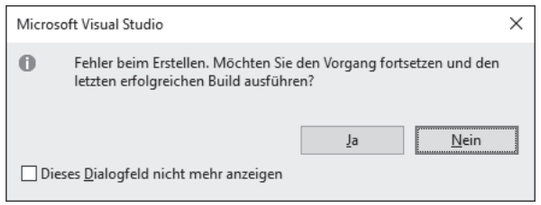

Abbildung 1.22 Fehler beim Erstellen

Betätigen Sie den Button NEIN, erscheint in Visual Studio die FEHLERLISTE mit der Beschreibung des Fehlers, siehe Abbildung 1.23.

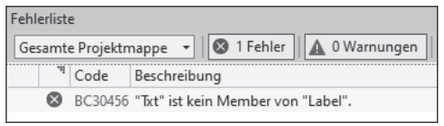

Abbildung 1.23 Beschreibung des Fehlers

Führen Sie einen Doppelklick auf der Zeile mit der Beschreibung des Fehlers aus, wird der fehlerhafte Code angezeigt und hervorgehoben, siehe Abbildung 1.24.

```
22      Private Sub CmdAnzeigen4_Click(sender As Object,
23          Dim x = 25, y = 17, z As Integer
24          z = x + y
25          MessageBox.Show("Das Ergebnis der " &
26              $"Berechnung:{vbCrLf}{x} + {y} = {z}")
27          LblAnzeige.Txt = "Ende"
28      End Sub
```

Abbildung 1.24 Hervorhebung des fehlerhaften Codes

In diesem Fall wurde der Name der Eigenschaft `Text` falsch geschrieben. Nach der Korrektur des Codes kann die Erstellung und Ausführung erneut gestartet werden.

1.8 Arbeiten mit Steuerelementen

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

1.8.1 Steuerelemente formatieren

Zur besseren Anordnung der Steuerelemente auf dem Formular können Sie diese 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 EIGENSCHAFTEN-Fenster 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 1.25). Das geschieht entweder

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

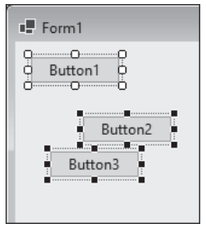

Abbildung 1.25 Mehrere markierte Elemente

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 1.26 zeigt ein Formular mit drei Buttons, die alle linksbündig ausgerichtet sind und den gleichen vertikalen Abstand zueinander haben.

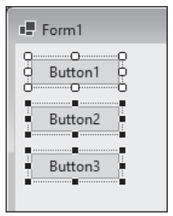

Abbildung 1.26 Nach der Formatierung

Übung

Laden Sie das Projekt *HalloWelt* aus Abschnitt 1.5, »Mein erstes Windows-Programm«, markieren Sie darin mehrere Steuerelemente, und testen Sie anschließend die einzelnen Möglichkeiten des FORMAT-Menüs aus.

1.8.2 Steuerelemente kopieren

Zur effektiveren Bearbeitung können vorhandene Steuerelemente einschließlich all 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 1.5 und kopieren Sie einzelne Steuerelemente. Kontrollieren Sie anschließend die Liste der vorhandenen Steuerelemente im EIGENSCHAFTEN-Fenster auf einheitliche Namensgebung.

1.8.3 Eigenschaften zur Laufzeit ändern

Steuerelemente haben die Eigenschaften `Size` (mit den Komponenten `Width` und `Height`) und `Location` (mit den Komponenten `X` und `Y`) zur Angabe von Größe und Position. `X` und `Y` geben die Koordinaten der oberen linken Ecke des Steuerelements an, gemessen von der oberen linken Ecke des umgebenden Elements (meist das Formular). Sämtliche Werte werden in Pixeln angegeben.

Alle diese 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 1.27).

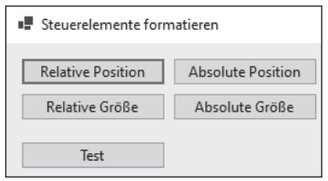

Abbildung 1.27 Position und Größe bestimmen

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

```
Private Sub CmdPositionRel_Click(...) Handles ...
    CmdTest.Location = New Point(
        CmdTest.Location.X + 20, CmdTest.Location.Y)
End Sub

Private Sub CmdPositionAbs_Click(...) Handles ...
    CmdTest.Location = New Point(100, 150)
End Sub

Private Sub CmdGroesseRel_Click(...) Handles ...
    CmdTest.Size = New Size(
        CmdTest.Size.Width + 20, CmdTest.Size.Height)
End Sub

Private Sub CmdGroesseAbs_Click(...) Handles ...
    CmdTest.Size = New Size(50, 100)
End Sub
```

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`. 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 PositionAbs 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 PositionRel 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`. 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 GroesseAbs wird die Größe des fünften Buttons auf die Werte `Width = 50` und `Height = 100` gestellt.
- Bei Betätigung des Buttons GroesseRel 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 1.28.

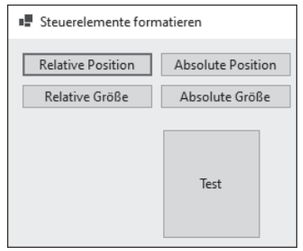

Abbildung 1.28 Veränderung von Eigenschaften zur Laufzeit

1.8.4 Ausgabe von Eigenschaften

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

```
Private Sub CmdAnzeigen_Click(...) Handles ...
    LblAnzeige.Text = $"X:{CmdTest.Location.X} " &     ' Ort, X-Koordinate
        $"Y:{CmdTest.Location.Y}{vbCrLf}" &            ' Ort, Y-Koordinate
        $"Breite:{CmdTest.Size.Width} " &              ' Breite
        $"Höhe:{CmdTest.Size.Height}"                  ' Höhe
End Sub
```

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.

Bei einer langen Anweisung, die sich über mehrere Zeilen erstreckt, können seit Visual Basic 2019 Kommentare am Ende jeder Zeile eingefügt werden.

Nach einigen Klicks und der Betätigung des Buttons CmdAnzeigen sieht das Formular aus wie in Abbildung 1.29.

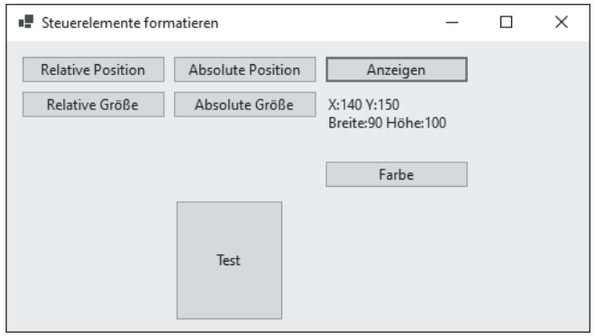

Abbildung 1.29 Anzeige der Eigenschaften

1.8.5 Farben und die Struktur »Color«

Die Hintergrundfarbe eines Steuerelements wird mit der Eigenschaft BackColor festgelegt. Dabei können Sie die Farbe zur Entwicklungszeit leicht mithilfe einer Farbpalette oder aus Systemfarben auswählen.

Ein Beispiel, ebenfalls im Projekt *GrundlagenSteuerelemente*:

```
Private Sub CmdFarbe_Click(...) Handles ...
    BackColor = Color.Yellow
    LblAnzeige.BackColor = Color.FromArgb(192, 255, 0)
End Sub
```

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. 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-, den Grün- und den Blau-Anteil der Farbe, jeweils zwischen 0 und 255.

Der betreffende Teil des Formulars sieht nach der Änderung der Farben aus wie in Abbildung 1.30.

Abbildung 1.30 Nach Änderung der Farben

Kapitel 2
Grundlagen

In diesem Kapitel erlernen Sie auf anschauliche Weise die Sprachgrundlagen von Visual Basic .NET 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 Visual Basic .NET 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 scharfe 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. Die Entwicklerin wählt den Datentyp danach aus, ob er Texte, Zahlen ohne Nachkommastellen, Zahlen mit Nachkommastellen oder zum Beispiel logische Werte speichern möchte. Außerdem muss sie sich bei Zahlen Gedanken über die Größe des Zahlenbereichs und die Genauigkeit machen.

Die wichtigsten von Visual Basic .NET 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 `Integer`, 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 `Single`, 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 `Boolean`, für Wahrheitswerte, also `True` oder `False` (wahr oder falsch)
- der Datentyp `Date` mit Werten für Datumsangaben vom 1. Januar des Jahres 1 bis zum 31. Dezember 9999
- 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 Sub CmdAnzeigen_Click(...) Handles ...
    ' Ganze Zahlen
    Dim by As Byte
    Dim sh As Short
    Dim it, bn, ok, hx As Integer
    Dim lg As Long

    ' Zahlen mit Nachkommastellen
    Dim si As Single
    Dim db, ex1, ex2 As Double
    Dim de As Decimal

    ' Boolesche Variable, Datum, Zeichen, Zeichenkette
    Dim bo As Boolean
    Dim dt As Date
    Dim ch As Char
    Dim st As String

    ' Ganze Zahlen
    by = 200
    sh = 30000
    it = 2_000_000_000
    lg = 3_000_000_000L
    bn = &B1001                ' oder: &B_10_01
    ok = &O173                 ' oder: &O_1_73
    hx = &H2F5                 ' oder: &H_2_F5

    ' Zahlen mit Nachkommastellen
    si = 1.0! / 7.123_456!
    db = 1.0 / 7.123_456
    de = 1D / 7.123_456D
    ex1 = 1.5E+17
    ex2 = 1.5E-17

    ' Boolesche Variable, Datum, Zeichen, Zeichenkette
    bo = True
    dt = "18.9.22 16:30"
    ch = "a"c
    st = "Hallo Welt"

    LblAnzeige.Text = $"Byte: {by}{vbCrLf}" &
        $"Short: {sh}{vbCrLf}" & $"Integer: {it}{vbCrLf}" &
        $"Long: {lg}{vbCrLf}" & $"(binäre Zahl): {bn}{vbCrLf}" &
```

```
        $"(oktale Zahl): {ok}{vbCrLf}" &
        $"(hexadezimale Zahl): {hx}{vbCrLf}{vbCrLf}" &
        $"Single: {si}{vbCrLf}" & $"Double: {db}{vbCrLf}" &
        $"Decimal: {de}{vbCrLf}{vbCrLf}" &
        $"Exponentialzahlen: {ex1} {ex2}{vbCrLf}" &
        $"Boolean: {bo}{vbCrLf}" & $"Date: {dt}{vbCrLf}" &
        $"Char: {ch}{vbCrLf}" & $"String: {st}"
End Sub
```

Listing 2.1 Projekt »GrundlagenDatentypen«

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

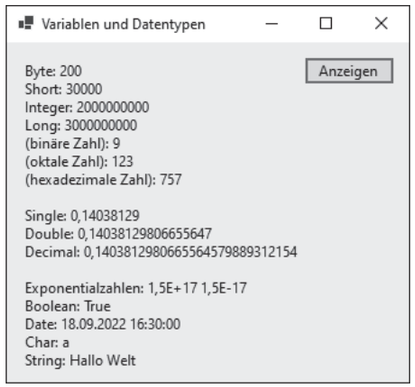

Abbildung 2.1 Wichtige Datentypen

Variablen können entweder mithilfe von `Dim <Name> As <Datentyp>` oder mithilfe von `Dim <Name> = <Wert>` deklariert werden. Im letzteren Fall erhält die Variable den Datentyp des zugewiesenen Werts. Auch die Kombination der beiden Schreibweisen ist möglich: `Dim <Name> As <Datentyp> = <Wert>`.

Mehrere Variablen können, durch Kommata getrennt, innerhalb einer Anweisung deklariert werden (zum Beispiel `Dim a, b As Integer, c, d As Double`). Dabei erhalten die beiden Variablen `a` und `b` den Datentyp `Integer` und die beiden Variablen `c` und `d` den Datentyp `Double`.

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

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.
- Das Zeichen `L` kennzeichnet eine Zahl als Wert des Datentyps `Long`.
- Wird einer `Integer`-Variablen ein Wert zugewiesen, der Nachkommastellen enthält, wird der Wert vor der Speicherung gerundet.
- Ganze Zahlen können auch in hexadezimaler Form zugewiesen werden, mithilfe von &H 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 &H2F5 entspricht der Dezimalzahl 757, denn es gilt: $2 \times 16^2 + 15 \times 16^1 + 5 \times 16^0 = 512 + 240 + 5 = 757$.
- Es kann auch eine oktale Zahl zugewiesen werden. Diese beginnen mit der Zeichenfolge &O, gefolgt von oktalen Ziffern, also den Ziffern von 0 bis 7. Ein Beispiel: Die Oktalzahl &O173 entspricht der Dezimalzahl 123, denn es gilt: $1 \times 8^2 + 7 \times 8^1 + 3 \times 8^0 = 64 + 56 + 3 = 123$.
- Eine weitere Möglichkeit zur Zuweisung bieten die binären Zahlen. Sie beginnen mit der Zeichenfolge &B, gefolgt von binären Ziffern, also 0 oder 1. Ein Beispiel: Die Binärzahl &B1001 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 Hexadezimal-, Oktal- oder Binärzahlen einsetzen.

Einige 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 (gemäß der lokalen Einstellung des Betriebssystems) ein Dezimalkomma dargestellt.
- Die Zuweisung einer zu großen Zahl führt zu einer Fehlermeldung, die Zuweisung einer zu kleinen Zahl zur Zuweisung des Werts 0.
- `Single`-Werte sollten mit dem Zeichen `!` gekennzeichnet werden, `Decimal`-Werte mit einem `D`.
- Sehr große oder sehr kleine Zahlen können auch mithilfe der Exponentialschreibweise zugewiesen werden. Zwei Beispiele: `1.5E+17` für `150000000000000000.0` oder `1.5E-17` für `0.000000000000000015`.
- 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, »Mathematische Funktionen«).

Werte für den Datentyp `Boolean` werden mit `True` und `False` zugewiesen und ausgegeben. Werte für einzelne Zeichen und für Zeichenketten werden in doppelten Anführungszeichen angegeben. Einzelne Zeichen müssen mit dem Zeichen `c` gekennzeichnet werden.

Der Name `Date` steht eigentlich nicht für einen Datentyp, sondern für eine Struktur. Ein Alias, also ein anderer Name für diese Struktur, ist `DateTime`. Mehr dazu in Abschnitt 6.2, »Datum und Uhrzeit«.

Von den hier genannten Datentypen kommen `Integer`, `Double`, `Boolean` und `String` am häufigsten zum Einsatz.

Übung »UDatentypen«

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

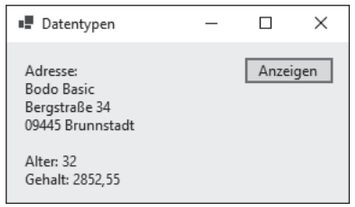

Abbildung 2.2 Übung »UDatentypen«

2.1.3 Gültigkeitsbereich

Eine Variable, die innerhalb einer Methode vereinbart wird, ist nur in der 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 derselben 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 Schlüsselwort `Private` deklariert werden. Damit ist sie außerhalb der Klasse unbekannt und ungültig. Wird sie dagegen mit dem Schlüsselwort `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 Kapitel 5.

Eine Objekteigenschaft muss mit Angabe eines Datentyps deklariert werden, auch wenn ihr unmittelbar ein Wert zugewiesen wird.

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 Class Form1
    Private Mx As Integer = 0

    Private Sub CmdAnzeigen1_Click(...) Handles ...
        Dim x = 0
        Mx += 1
        x += 1
        LblAnzeige.Text = $"x: {x} Mx: {Mx}"
    End Sub

    Private Sub CmdAnzeigen2_Click(...) Handles ...
        Dim Mx = 0
        Mx += 1
        LblAnzeige.Text = $"Mx: {Mx}"
    End Sub
End Class
```

Listing 2.2 Projekt »GrundlagenGueltigkeit«

In der ersten Methode wird der Wert der Objekteigenschaft `Mx` bei jedem Aufruf erhöht. Die Zuweisung `Mx += 1` entspricht der Zuweisung `Mx = Mx + 1`, siehe auch Abschnitt 2.2.1, »Rechenoperatoren«. Ebenso entspricht die Zuweisung `x += 1` 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.

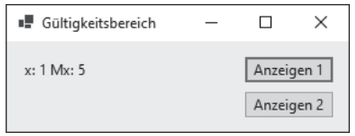

Abbildung 2.3 Lokale Variable »x« und Objekteigenschaft »Mx«

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.

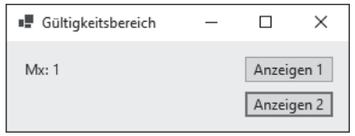

Abbildung 2.4 Lokale Variable »Mx«

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 Mx As Integer = 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).

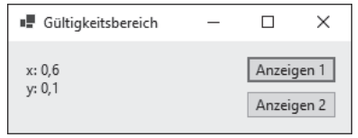

Abbildung 2.5 Ausgabe der ersten Methode nach einigen Klicks

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

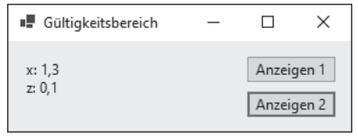

Abbildung 2.6 Ausgabe der zweiten Methode nach weiteren Klicks

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, wie zum Beispiel `vbCrLf`. 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 Class Form1
    Private Const MaxWert = 75
    Private Const Eintrag = "Picture"
    ...
    Private Sub CmdAnzeigen1_Click(...) Handles ...
```

```
        Const MaxWert = 55
        Const MinWert = 5
        LblAnzeige.Text = $"{(MaxWert - MinWert) / 2}{vbCrLf}{Eintrag}"
    End Sub
    ...
End Class
```

Listing 2.3 Projekt »GrundlagenKonstanten«, Konstanten

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.

Abbildung 2.7 Konstanten

Sowohl für konstante Objekteigenschaften als auch für lokale Konstanten gilt: Sie werden mithilfe des Schlüsselworts `Const` definiert und müssen bei ihrer Deklaration einen Wert erhalten. Es kann ein Datentyp angegeben werden. Ist das nicht der Fall, ergibt sich der Datentyp aus dem zugewiesenen Wert.

Mithilfe der String-Interpolation 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.

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 den Elementen einer Enumeration Werte zugewiesen. Für Visual Basic .NET gibt es zahlreiche integrierte 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 Class Form1
    ...
    Private Enum Farbe As Integer
        Rot = 1
        Gelb = 2
        Blau = 3
    End Enum
    ...
    Private Sub CmdAnzeigen2_Click(...) Handles ...
        LblAnzeige.Text = $"Farbe: {Farbe.Gelb} {Convert.ToInt32(Farbe.Gelb)}"
    End Sub

    Private Sub CmdAnzeigen3_Click(...) Handles ...
        LblAnzeige.Text =
            $"Sonntag: {DayOfWeek.Sunday} " &
            $"{Convert.ToInt32(DayOfWeek.Sunday)}{vbCrLf}" &
            $"Samstag: {DayOfWeek.Saturday} " &
            $"{Convert.ToInt32(DayOfWeek.Saturday)}"
    End Sub
End Class
```

Listing 2.4 Projekt »GrundlagenKonstanten«, Enumerationen

Mithilfe des Schlüsselworts `Enum` wird die Enumeration `Farbe` vom Datentyp `Integer` 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, siehe Abbildung 2.8.

Die Umwandlung wird mithilfe der statischen Methode `ToInt32()` der Klasse `Convert` vorgenommen. Eine Integer-Variable belegt einen Speicherplatz von 4 Byte = 32 Bit, daher der Name der Methode. Die Klasse `Convert` bietet viele weitere Methoden (also Konvertierung) zur Umwandlung von Werten.

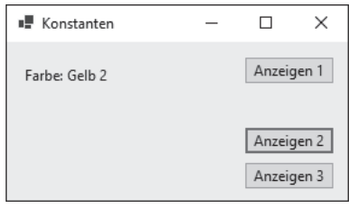

Abbildung 2.8 Erste Enumeration

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.

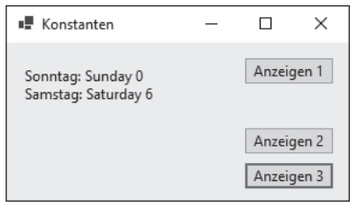

Abbildung 2.9 Zweite Enumeration

2.2 Operatoren

Zum Zusammensetzen von Ausdrücken werden in Visual Basic .NET 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
\	Ganzzahldivision
Mod	Modulo
^	Potenzierung

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.

Die Ganzzahldivision mithilfe des Operators `\` erfolgt in zwei Schritten. Im ersten Schritt werden Dividend und Divisor einzeln gerundet. Im zweiten Schritt werden die beiden verbliebenen Zahlen geteilt, anschließend werden die Ziffern nach dem Komma abgeschnitten. Der Modulo-Operator `Mod` berechnet den Rest einer Division. Zur Potenzierung einer Zahl dient der Operator `^` (hoch).

Einige Beispiele für diese drei Operatoren sehen Sie in Tabelle 2.2. Im Projekt *GrundlagenOperatorenRechnen* werden diese Beispiele nachvollzogen.

Ausdruck	Ergebnis	Erläuterung
`23 \ 4`	5	Das mathematische Ergebnis ist 5,75, davon werden die Nachkommastellen abgeschnitten
`19.3 \ 4.2`	4	Die Rundung ergibt 19 \ 4, das mathematische Ergebnis ist 4,75, davon werden die Nachkommastellen abgeschnitten

Tabelle 2.2 Modulo-Operator

Ausdruck	Ergebnis	Erläuterung
`19 Mod 4`	3	19 durch 4 ist 4 Rest 3
`19.3 Mod 4.2`	2.5	19,3 durch 4,2 ist 4 Rest 2,5
`2 ^ 5`	32	2 × 2 × 2 × 2 × 2 = 32
`-2.1 ^ -5.4`	–0,018...	Das entspricht 1 / (–2,1 hoch 5,4)

Tabelle 2.2 Modulo-Operator (Forts.)

Übung »UOperatorenRechnen«

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

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

2.2.2 Vergleichsoperatoren

Mithilfe von Vergleichsoperatoren können Sie feststellen, ob bestimmte Bedingungen zutreffen oder nicht. Sie können zum Vergleich von zwei Zahlen oder zwei Zeichenketten genutzt werden, siehe Tabelle 2.3. Das Ergebnis wird zur Ablaufsteuerung von Programmen eingesetzt. In Abschnitt 2.4, »Verzweigungen mit ›If‹ und ›IIf()‹«, werde ich hierauf noch genauer eingehen.

Für Zeichenketten dient der Operator `Like` zum Mustervergleich. Dabei können Sie u. a. die Platzhalter `*` (beliebig viele beliebige Zeichen) und `?` (genau ein beliebiges Zeichen) einsetzen.

Operator	Beschreibung	Anwendung
`<`	kleiner als	Zahlen
`<=`	kleiner als oder gleich	Zahlen
`>`	größer als	Zahlen
`>=`	größer als oder gleich	Zahlen
`=`	gleich	Zahlen / Zeichenketten

Tabelle 2.3 Vergleichsoperatoren

Operator	Beschreibung	Anwendung
`<>`	ungleich	Zahlen / Zeichenketten
`Like`	Mustervergleich	Zeichenketten

Tabelle 2.3 Vergleichsoperatoren (Forts.)

Einige Beispiele für diese Operatoren sehen Sie in Tabelle 2.4. Im Projekt *Grundlagen-OperatorenVergleich* werden diese Beispiele nachvollzogen.

Ausdruck	Ergebnis	Erläuterung
`5 > 3`	`True`	5 ist größer als 3.
`3 = 3.2`	`False`	3 ist nicht gleich 3,2.
`5 + 3 * 2 >= 12`	`False`	11 ist nicht größer als 12.
`"Maier" <> "Mayer"`	`True`	Der dritte Buchstabe unterscheidet sich.
`"abxba" Like "a*a"`	`True`	Zwischen den beiden `"a"` stehen beliebig viele beliebige Zeichen.
`"abxba" Like "a?a"`	`False`	Zwischen den beiden `"a"` steht nicht nur ein beliebiges Zeichen.
`"aba" Like "a?a"`	`True`	Zwischen den beiden `"a"` steht genau ein beliebiges Zeichen.
`"asdlfigc" Like "a?d?f*c"`	`True`	Zwischen `"a"` und `"d"` sowie zwischen `"d"` und `"f"` steht jeweils genau ein beliebiges Zeichen, zwischen `"f"` und `"c"` stehen beliebig viele beliebige Zeichen.

Tabelle 2.4 Einsatz von Vergleichsoperatoren

Übung »UOperatorenVergleich«

Ermitteln Sie im Projekt *UOperatorenVergleich* die Ergebnisse der beiden folgenden Ausdrücke, speichern Sie sie in Variablen eines geeigneten Datentyps, und zeigen Sie sie an:

- 1. Ausdruck: `12 - 3 >= 4 * 2.5`
- 2. Ausdruck: `"Maier" Like "M??er"`

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 2.5.

Operator	Beschreibung	Das Ergebnis ist True, wenn ...
Not	Nicht	... der Ausdruck False ist.
And	Und	... beide Ausdrücke True sind.
Or	inklusives Oder	... mindestens ein Ausdruck True ist.
Xor	exklusives Oder	... genau ein Ausdruck True ist.

Tabelle 2.5 Logische Operatoren

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

Ausdruck	Ergebnis
Not A < B	False
B > A And C > B	True
B < A Or C < B	False
B < A Xor C > B	True

Tabelle 2.6 Ausdrücke mit logischen Operatoren

Alle Operationen werden im Projekt *GrundlagenOperatorenLogisch* nachvollzogen.

Übung »UOperatorenLogisch«

Ermitteln Sie im Projekt *UOperatorenLogisch* die Ergebnisse der beiden folgenden Ausdrücke, speichern Sie sie in Variablen eines geeigneten Datentyps, und zeigen Sie sie an:

- 1. Ausdruck: 4 > 3 And -4 > -3
- 2. Ausdruck: 4 > 3 Or -4 > -3

2.2.4 Zuweisungsoperatoren

Neben dem Zuweisungsoperator = gibt es zur Verkürzung von Anweisungen die kombinierten Zuweisungsoperatoren (siehe Tabelle 2.7), 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.
\=	x \= 3	x wird durch 3 geteilt, die Nachkommastellen werden abgeschnitten.
^=	x ^= 3	Der Wert von x wird hoch 3 gerechnet.
&=	z &= "abc"	Die Zeichenkette z wird um den Text »abc« verlängert.

Tabelle 2.7 Zuweisungsoperatoren

Alle Operationen werden im Projekt *GrundlagenVerbundzuweisungen* nachvollzogen.

2.2.5 Rangfolge der Operatoren

Enthält ein Ausdruck mehrere Operatoren, werden die einzelnen Teilausdrücke gemäß der Rangfolge (= Priorität) der Operatoren ausgewertet und aufgelöst, siehe Tabelle 2.8. Je weiter oben die Operatoren in der Tabelle stehen, desto höher ist ihre Rangfolge.

Operator	Beschreibung
^	Exponentialoperator
+ -	Positives und negatives Vorzeichen
* /	Multiplikation, Division
\	Ganzzahldivision
Mod	Modulo

Tabelle 2.8 Priorität der Operatoren

Operator	Beschreibung
+ -	Addition, Subtraktion
&	Verkettung
= <> < > <= >= Like	Vergleichsoperatoren (= nicht für Zuweisung)
Not	Logisches Nicht
And	Logisches Und
Or	Logisches Oder
Xor	Logisches exklusives Oder

Tabelle 2.8 Priorität der Operatoren (Forts.)

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 2.9 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 And b<>10
2	a=5 b=10	a>0 Or b<>10
3	z=10 w=100	z<>0 Or z>w Or w-z=90
4	z=10 w=100	z=11 And z>w Or w-z=90
5	x=1.0 y=5.7	x>=0.9 And y<=5.8
6	x=1.0 y=5.7	x>=0.9 And Not y<=5.8
7	n1=1 n2=17	n1>0 And n2>0 Or n1>n2 And n2<>17
8	n1=1 n2=17	n1>0 And (n2>0 Or n1>n2) And n2<>17

Tabelle 2.9 Übung »UOperatorenRangfolge«

2.3 Einfache Steuerelemente

Die Windows-Programmierung mit Visual Basic .NET innerhalb von Visual Studio besteht prinzipiell aus zwei Teilen: der Arbeit mit den visuellen Steuerelementen und der Programmierung mit der Sprache Visual Basic .NET. 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«

Ein *Panel* dient normalerweise als Container für andere Steuerelemente. 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 2.10 sehen Sie das Panel im Startzustand und in Abbildung 2.11 nach einigen Klicks.

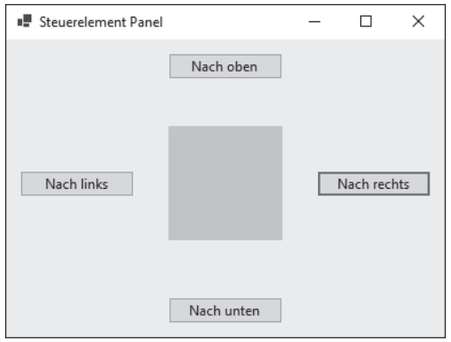

Abbildung 2.10 Zustand zu Beginn

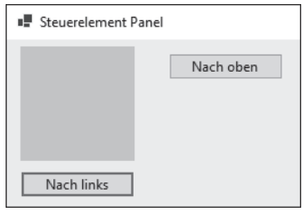

Abbildung 2.11 Zustand nach einigen Klicks

Der Programmcode:

```
Private Sub CmdNachOben_Click(...) Handles ...
    P.Location = New Point(P.Location.X, P.Location.Y - 10)
End Sub

Private Sub CmdNachLinks_Click(...) Handles ...
    P.Location = New Point(P.Location.X - 10, P.Location.Y)
End Sub

Private Sub CmdNachRechts_Click(...) Handles ...
    P.Location = New Point(P.Location.X + 10, P.Location.Y)
End Sub

Private Sub CmdNachUnten_Click(...) Handles ...
    P.Location = New Point(P.Location.X, P.Location.Y + 10)
End Sub
```

Listing 2.5 Projekt »SteuerelementPanel«

2.3.2 Steuerelement »Timer«

Ein Zeitgeber (engl. *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. Er ist nicht im Formular sichtbar und wird stattdessen unterhalb des Formulars angezeigt. Auch zur Laufzeit ist er nicht sichtbar. Seine wichtigste Eigenschaft ist das Zeitintervall in Millisekunden, in dem das Ereignis auftritt.

Die Eigenschaft `Enabled` dient der Aktivierung beziehungsweise Deaktivierung eines Steuerelements, hier des Zeitgebers. Sie können sie zur Entwicklungszeit oder zur Laufzeit auf `True` oder `False` stellen.

Im nachfolgenden Programm im Projekt *SteuerelementTimer* 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 2.12). Dies wird durch den Timer gesteuert, bei dem der Wert für die Eigenschaft `Interval` auf 500 steht. Nach Betätigung des STOP-Buttons kommt kein weiteres x mehr hinzu.

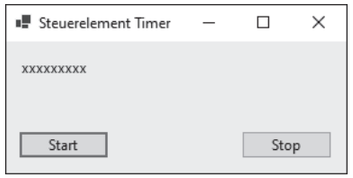

Abbildung 2.12 Zustand einige Sekunden nach dem Start

Der zugehörige Code:

```
Private Sub CmdStart_Click(...) Handles ...
    TimAnzeige.Enabled = True
End Sub

Private Sub CmdStop_Click(...) Handles ...
    TimAnzeige.Enabled = False
End Sub

Private Sub TimAnzeige_Tick(...) Handles TimAnzeige.Tick
    LblAnzeige.Text &= "x"
End Sub
```

Listing 2.6 Projekt »SteuerelementTimer«

Übung »UPanelTimer«

Erstellen Sie im Projekt *UPanelTimer* eine Windows-Anwendung. In der Mitte eines Formulars sollen zu Beginn vier Panels verschiedener Farbe der Größe 20 × 20 Pixel platziert werden (siehe Abbildung 2.13).

Sobald der 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 2.14).

Abbildung 2.13 Zustand zu Beginn

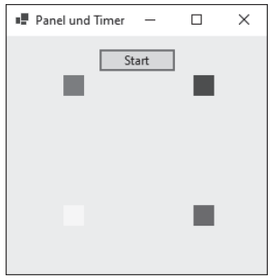

Abbildung 2.14 Zustand einige Sekunden nach dem Start

Übung »UKran«

Diese Übung gehört nicht zum Pflichtprogramm. Sie ist etwas umfangreicher, verdeutlicht aber die Möglichkeiten einer schnellen Visualisierung von Prozessen durch Visual Basic .NET innerhalb von Visual Studio durch einige wenige Programmzeilen.

Konstruieren Sie im Projekt *UKran* aus mehreren Panels einen Kran (Fundament, senkrechtes Hauptelement, waagerechter Ausleger, senkrechter Haken am Ausleger). Die Benutzer sollen die Möglichkeit haben, über insgesamt acht Buttons die folgenden Aktionen auszulösen:

- Haken um zehn Pixel ausfahren/einfahren
- Ausleger um zehn Pixel ausfahren/einfahren
- Kran um zehn Pixel nach rechts/links fahren
- Kran um zehn Pixel in der Höhe ausfahren/einfahren

Denken Sie daran, dass bei vielen Bewegungen mehrere Steuerelemente bewegt werden müssen, da der Kran sonst seinen Zusammenhalt verliert. Die Aktionen resultieren aus Änderungen der Größe oder des Orts oder beidem.

In Abbildung 2.15 sehen Sie den Kran im Startzustand und in Abbildung 2.16 nach einigen Klicks.

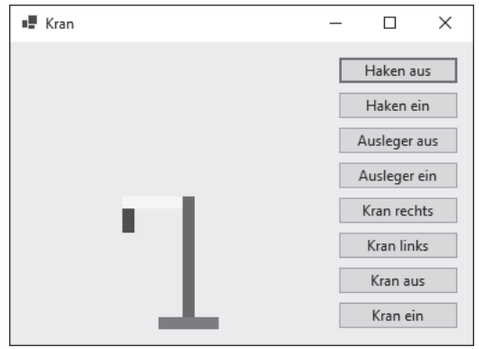

Abbildung 2.15 Übung »UKran«, Zustand zu Beginn

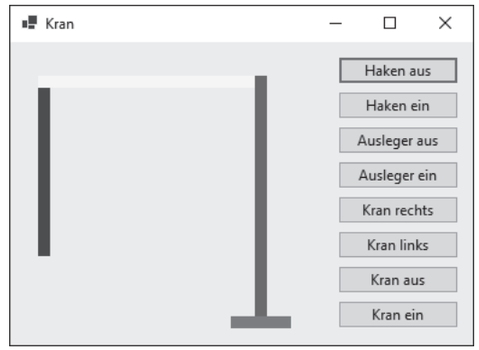

Abbildung 2.16 Übung »UKran«, Zustand nach einigen Aktionen

2.3.3 Steuerelement »TextBox«

Eine *TextBox* (deutsch: Textfeld oder Texteingabefeld) dient in erster Linie dazu, die Eingabe von Text oder Zahlen vom Benutzer entgegenzunehmen. Diese Eingaben werden in der Eigenschaft `Text` der TextBox gespeichert. Das Aussehen und das Verhalten einer TextBox können durch weitere Eigenschaften bestimmt werden:

- `Multiline`: Steht `Multiline` auf `True`, können Sie bei der Eingabe und bei der Anzeige mit mehreren Textzeilen arbeiten.
- `ScrollBars`: Sie können eine TextBox mit vertikalen und/oder horizontalen Bildlaufleisten zur Bearbeitung längerer Texte versehen.

- `MaxLength`: Damit lässt sich die Anzahl der Zeichen der TextBox beschränken. Ist keine Beschränkung vorgesehen, kann die TextBox insgesamt 32.768 Zeichen aufnehmen.
- `PasswordChar`: Haben Sie für diese Eigenschaft im Entwurfsmodus ein Platzhalterzeichen eingegeben, wird während der Laufzeit für jedes eingegebene Zeichen dieser Platzhalter angezeigt. Diese Eigenschaft wird vor allem bei Passwortabfragen verwendet.

Der Inhalt einer TextBox kann von Benutzern mit den gewohnten Mitteln (zum Beispiel Strg + C und Strg + V) in die Zwischenablage kopiert beziehungsweise aus der Zwischenablage eingefügt werden.

Benötigen Sie umfangreichere Möglichkeiten zur Formatierung, zum Beispiel zur Tief- oder Hochstellung einzelner Zeichen, empfehle ich das Steuerelement *RichTextBox*, siehe Abschnitt 7.8.

Im nachfolgenden Programm im Projekt *SteuerelementTextBox* können Benutzer in einer TextBox einen Text eingeben. Nach Betätigung des Buttons Anzeigen wird der eingegebene Text ausgegeben (siehe Abbildung 2.17).

Abbildung 2.17 Eingabe in TextBox

Der Code lautet wie folgt:

```
Private Sub CmdAnzeigen_Click(...) Handles ...
    LblAnzeige.Text = $"Ihre Eingabe: {TxtEingabe.Text}"
End Sub
```

Listing 2.7 Projekt »SteuerelementTextBox«, Eingabe von Text

In der Eigenschaft `Text` der TextBox wird die Eingabe gespeichert. Der Wert der Eigenschaft wird in einen längeren Ausgabetext eingebettet.

Bei der Eingabe und Auswertung von Zahlen sind einige Besonderheiten zu beachten. Im nachfolgenden Programm, ebenfalls im Projekt *SteuerelementTextBox*, können Benutzer in einer TextBox eine Zahl eingeben. Nach Betätigung des Buttons Rechnen

wird der Wert dieser Zahl verdoppelt, das Ergebnis wird in einem Label darunter ausgegeben:

```
Private Sub CmdRechnen_Click(...) Handles ...
    Dim wert As Double
    wert = Convert.ToDouble(TxtEingabe.Text)
    wert *= 2
    LblAnzeige.Text = $"Ergebnis: {wert}"
End Sub
```

Listing 2.8 Projekt »SteuerelementTextBox«, Eingabe von Zahlen

Der Inhalt der TextBox soll explizit in eine Zahl (mit möglichen Nachkommastellen) umgewandelt werden. Das erreichen Sie mithilfe der Methode `ToDouble()` aus der Klasse `Convert`.

Enthält die eingegebene Zeichenkette eine Zahl, wird sie in diese Zahl umgewandelt. Anschließend kann mit dieser Zahl gerechnet werden.

Enthält die eingegebene Zeichenkette keine Zahl, kommt es zu einem Laufzeitfehler. Diese Situation sollten Sie vermeiden. Sie können zum Beispiel zuvor prüfen, ob es sich bei der Zeichenkette um eine gültige Zahl handelt, und entsprechend reagieren. Das wird Ihnen möglich sein, sobald Sie die in Abschnitt 2.4, »Verzweigungen mit ›If‹ und ›IIf()‹«, beschriebenen Verzweigungen zur Programmsteuerung beherrschen.

Allgemein können Sie Programme so schreiben, dass ein Programmabbruch abgefangen wird. Dazu werden Sie in der Lage sein, sobald Sie die Ausnahmebehandlung (siehe hierzu Abschnitt 3.4, »Laufzeitfehler und Exception Handling«) anwenden können.

Es folgen einige Eingabebeispiele. In Abbildung 2.18 sehen Sie das Ergebnis für die korrekte Eingabe einer Zahl mit Stellen nach einem Dezimalkomma.

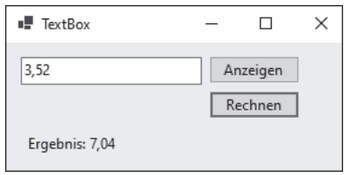

Abbildung 2.18 Korrekte Eingabe mit Dezimalkomma

Die Eingabe einer Zeichenkette, wie zum Beispiel »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 2.19).