7. Auflage

Thomas Theis

Einstieg in Visual Basic 2019

Ideal für Programmieranfänger

- Schnell erste eigene Windows-Programme entwickeln
- Alle Grundlagen zu GUIs, Datenbank- und Internetanwendungen
- Inkl. Einführung in Windows Presentation Foundation

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. Sie sind anschließend in der Lage, Ihr erstes eigenes Windows-Programm zu erstellen.

Bei Visual Basic .NET handelt es sich um eine objektorientierte Programmiersprache, die im Zusammenhang mit dem .NET Framework von Microsoft zur Verfügung gestellt wird. Mithilfe der Entwicklungsumgebung Visual Studio, die es aktuell in der Version 2019 gibt, können Sie in mehreren Sprachen, u. a. in Visual Basic .NET, programmieren.

Die Ursprünge von Visual Basic .NET liegen in der Programmiersprache BASIC, die erstmals im Jahre 1964 erschien. Die Programmiersprache Visual Basic wurde im Jahre 1991 von Microsoft gemeinsam mit einer integrierten Entwicklungsumgebung eingeführt. Sie ermöglicht u. a. die einfache Erstellung von Anwendungen mit Benutzeroberflächen. Im Jahre 2002 führte Microsoft die völlig neu entwickelte, objektorientierte Programmiersprache Visual Basic .NET ein.

Das .NET Framework bietet Klassenbibliotheken, Programmierschnittstellen und Dienstprogramme zur Entwicklung von Anwendungen. Außerdem wird eine Laufzeitumgebung zur Ausführung der Anwendungen zur Verfügung gestellt.

Sie können die frei verfügbare Community-Version der Entwicklungsumgebung Visual Studio 2019 nutzen. Sie lässt sich bei Microsoft über *https://visualstudio.microsoft.com/de/vs* herunterladen und auf Ihrem Windows-PC installieren. Eine Installationsanleitung finden Sie in Anhang A. Die Entwicklungsumgebung umfasst einen Editor zur Erstellung des Programmcodes, einen Compiler zur Erstellung der ausführbaren Programme, einen Debugger zur Fehlersuche und vieles mehr.

Mit Visual Basic .NET und dem Visual Studio lassen sich Anwendungen unterschiedlichen Typs erstellen, das sind u. a.:

- klassische Windows Forms-Anwendungen mit einfach zu erstellenden grafischen Benutzeroberflächen und ereignisorientierter Programmierung
- Moderne WPF-Anwendungen mit XAML. Dabei werden die Klassenbibliothek Windows Presentation Foundation (WPF) und die Auszeichnungssprache eXtensible Application Markup Language (XAML) genutzt.
- Datenbankanwendungen mit lesendem und schreibendem Zugriff auf viele unterschiedliche Datenbanksysteme. Für den Zugriff auf relationale Datenbanken kann man mit dem ADO.NET Framework als Teil des .NET Frameworks arbeiten.
- Dynamische Internetanwendungen, bei denen die Webseiten als interaktive Benutzeroberflächen dienen. Dabei wird das ASP.NET Framework als Teil des .NET Frameworks genutzt. ASP steht für Active Server Pages.

1.1 Aufbau dieses Buchs

Beispiele

Dieses Buch vermittelt Ihnen zunächst einen einfachen Einstieg in die Programmierung mit Visual Basic und Visual Studio 2019. Die Bearbeitung der Beispiele und das selbstständige Lösen der vorliegenden Übungsaufgaben helfen Ihnen dabei, da sie für schnelle Erfolgserlebnisse sorgen, die Sie zum Weitermachen motivieren sollen. In späteren Kapiteln werden auch die komplexen Themen vermittelt.

Grundlagen

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 kennen, werden gemeinsam vermittelt. Die Anschaulichkeit einer Windows-Anwendung hilft Ihnen dabei, den eher theoretischen Hintergrund der Programmiersprache leichter zu verstehen.

Die Projekte in diesem Buch wurden unter Windows 10 bearbeitet. Auch die Screenshots sind unter dieser Windows-Version entstanden.

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

1.2 Mein erstes Windows-Programm

Anhand eines ersten Projekts werden die Schritte durchlaufen, die zur Erstellung eines einfachen Programms mithilfe von Visual Basic notwendig sind. Das Programm soll nach dem Aufruf zunächst aussehen wie in Abbildung 1.1. Nach Betätigung des Buttons HALLO soll sich der Text in der obersten Zeile verändern (siehe Abbildung 1.2).

Abbildung 1.1 Erstes Programm, nach dem Aufruf

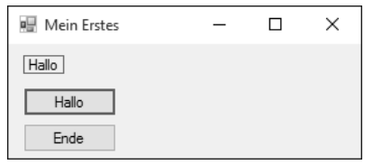

Abbildung 1.2 Nach Betätigung des Buttons »Hallo«

1.3 Visual Basic-Entwicklungsumgebung

Während der Projekterstellung lernen Sie Schritt für Schritt die Visual Studio-Entwicklungsumgebung für Visual Basic kennen.

1.3.1 Ein neues Projekt

Nach dem Aufruf des Programms *Visual Studio Community 2019* können Sie auf dem Startbildschirm rechts die Schaltfläche NEUES PROJEKT ERSTELLEN auswählen. Sollte der Startbildschirm nicht sichtbar sein, können Sie auch den Menüpunkt DATEI • NEU • PROJEKT ausführen.

In beiden Fällen gelangen Sie zum Dialogfeld NEUES PROJEKT ERSTELLEN. Auf der rechten Seite sehen Sie eine längere Liste von Projektvorlagen. Wählen Sie die Vorlage VB WINDOWS FORMS-APP (.NET FRAMEWORK) aus (siehe Abbildung 1.3). Ab dem zweiten Projekt wird diese Vorlage auch auf der linken Seite erscheinen, unter ZULETZT VERWENDETE PROJEKTVORLAGEN.

Abbildung 1.3 Projektvorlage VB Windows Forms-App

Nach Betätigung des Buttons WEITER tragen Sie den Projektnamen ein, hier: *MeinErstes*. Achten Sie darauf, dass das Häkchen gesetzt ist bei: LEGEN SIE DIE PROJEKTMAPPE UND DAS PROJEKT IM SELBEN VERZEICHNIS AB.

Nach Betätigung des Buttons ERSTELLEN erscheinen einige Elemente der Entwicklungsumgebung. Folgende Elemente sind besonders wichtig:

Form

- Das Benutzerformular (engl.: *Form*) enthält die Oberfläche für den Benutzer des Programms (siehe Abbildung 1.4).

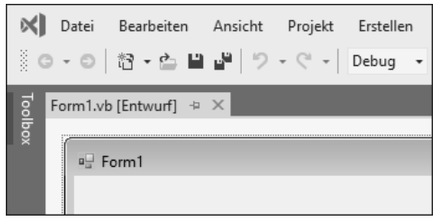

Abbildung 1.4 Benutzerformular

- Der WERKZEUGKASTEN (engl.: *Toolbox*) 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 1.5). Sollten in der Toolbox keine Steuerelemente angezeigt werden, klicken Sie einmal auf das Benutzerformular und anschließend wieder auf die Toolbox.

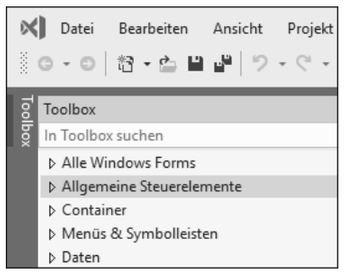

Abbildung 1.5 Werkzeugkasten mit Kategorien von Steuerelementen

- Das EIGENSCHAFTEN-Fenster (engl.: *Properties Window*) dient zum Anzeigen und Ändern der Eigenschaften von Steuerelementen innerhalb des Formulars durch den Programmentwickler (siehe Abbildung 1.6). Ich empfehle Ihnen, sich die Eigenschaften in alphabetischer Reihenfolge anzeigen zu lassen. Dazu einfach das zweite Symbol von links, unter FORM1, betätigen.

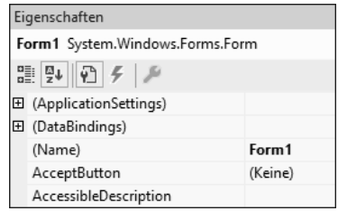

Abbildung 1.6 Eigenschaften-Fenster

- Der PROJEKTMAPPEN-EXPLORER (engl.: *Solution Explorer*) zeigt das geöffnete Projekt und die darin vorhandenen Elemente (siehe Abbildung 1.7).

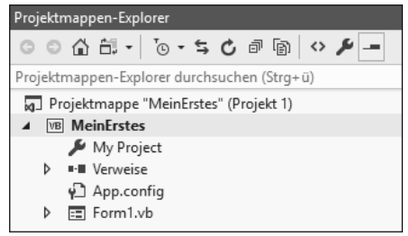

Abbildung 1.7 Projektmappen-Explorer

Elemente anzeigen

Sollten der WERKZEUGKASTEN, das EIGENSCHAFTEN-Fenster oder der PROJEKTMAPPEN-EXPLORER nicht angezeigt werden, können Sie das betreffende Element über das Menü ANSICHT einblenden. Sollte das Formular einmal nicht sichtbar sein, können Sie es über einen Doppelklick auf den Namen (FORM1.VB) im PROJEKTMAPPEN-EXPLORER einblenden. Sollten die Eigenschaften eines Steuerelements nicht im bereits sichtbaren EIGENSCHAFTEN-Fenster angezeigt werden, markieren Sie zunächst den Namen der Formulardatei (FORM1.VB) im PROJEKTMAPPEN-EXPLORER und anschließend das betreffende Steuerelement.

Zunächst werden nur einfache Programme mit wenigen Elementen geschrieben, daher benötigen Sie den PROJEKTMAPPEN-EXPLORER noch nicht. Es empfiehlt sich, das EIGENSCHAFTEN-Fenster nach oben zu vergrößern.

1.3.2 Einfügen von Steuerelementen

Label, Button

Zunächst sollen drei Steuerelemente in das Formular eingefügt werden: ein Bezeichnungsfeld (Label) und zwei Befehlsschaltflächen (Buttons). Ein Bezeichnungsfeld dient im Allgemeinen dazu, feste oder veränderliche Texte auf der Benutzeroberfläche anzuzeigen. In diesem Programm soll das Label einen Text anzeigen. Ein Button dient zum Starten bestimmter Programmteile oder, allgemeiner ausgedrückt, zum Auslösen von Ereignissen. In diesem Programm sollen die Buttons den Text anzeigen bzw. das Programm beenden.

Allgemeine Steuerelemente

Um ein Steuerelement einzufügen, ziehen Sie es mithilfe der Maus vom Werkzeugkasten an die gewünschte Stelle im Formular. Alle Steuerelemente finden sich im Werkzeugkasten unter Alle Windows Forms. Übersichtlicher ist der Zugriff über Allgemeine Steuerelemente (engl.: *Common Controls*) (siehe Abbildung 1.8).

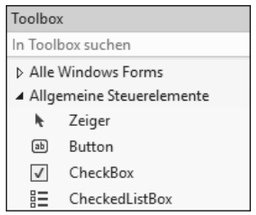

Abbildung 1.8 Kategorie »Allgemeine Steuerelemente«

Steuerelement auswählen

Ein Doppelklick auf ein Steuerelement im Werkzeugkasten fügt es ebenfalls in die Form ein. Anschließend können noch Ort und Größe verändert werden. Dazu müssen Sie das betreffende Steuerelement vorher durch Anklicken ausgewählt haben (siehe Abbildung 1.9). Ein überflüssiges Steuerelement können Sie durch Auswählen und Drücken der Taste Entf entfernen.

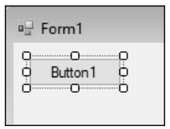

Abbildung 1.9 Ausgewählter Button

Die Größe und andere Eigenschaften des Formulars selbst können auch verändert werden. Dazu müssen Sie es vorher durch Anklicken einer freien Stelle auswählen.

1.3.3 Arbeiten mit dem Eigenschaften-Fenster

Die eingefügten Steuerelemente haben zunächst einheitliche Namen und Aufschriften, diese sollten allerdings zur einfacheren Programmentwicklung geändert werden. Es gibt bestimmte Namenskonventionen, die die Lesbarkeit erleichtern: Die Namen beinhalten den Typ (mit drei Buchstaben abgekürzt) und die Aufgabe des Steuerelements (jeweils mit großem Anfangsbuchstaben). Aus den Namen der Steuerelemente ergeben sich auch die Namen der sogenannten Ereignisprozeduren, ebenfalls mit großem Anfangsbuchstaben (siehe Abschnitt 1.3.5). Seit Visual Studio 2019 wird stärker auf die Einhaltung der Namenskonventionen geachtet.

Cmd, Txt, Lbl ...

Ein Button (eigentlich: Command Button), der die Anzeige der Zeit auslösen soll, wird beispielsweise mit `CmdZeit` bezeichnet. Weitere Vorsilben sind `Txt` (Textfeld/TextBox), `Lbl` (Bezeichnungsfeld/Label), `Opt` (Optionsschaltfläche/RadioButton), `Frm` (Formular/Form) und `Chk` (Kontrollkästchen/CheckBox).

Zur Änderung des Namens eines Steuerelements müssen Sie es zunächst auswählen. Die Auswahl kann entweder durch Anklicken des Steuerelements auf dem Formular oder durch Auswahl aus der Liste am oberen Ende des Eigenschaften-Fensters geschehen.

Eigenschaften-Fenster

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 Spalte ihr aktueller Wert. Die Eigenschaft *(Name)* steht am Anfang der Liste der Eigenschaften. Die betreffende Zeile wird durch Anklicken ausgewählt, und der neue Name wird eingegeben. Nach Bestätigung mit der Taste ↵ ist die Eigenschaft geändert (siehe Abbildung 1.10).

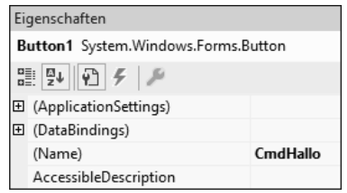

Abbildung 1.10 Button, Eigenschaft »Name« geändert

Eigenschaft Text

Die Aufschrift von Buttons, Labels und Formularen ist in der Eigenschaft `Text` angegeben. Sobald diese Eigenschaft verändert wird, erscheint die ver-

änderte Aufschrift in dem betreffenden Steuerelement. Auch die Aufschrift und die Größe des Formulars sollten Sie mithilfe der Eigenschaft `Size` ändern. Im Folgenden sind die gewünschten Eigenschaften für die Steuerelemente dieses Programms in Tabellenform angegeben (siehe Tabelle 1.1).

Startzustand

Zu diesem Zeitpunkt legen Sie den Startzustand fest, also die Eigenschaften, die die Steuerelemente zu Beginn des Programms bzw. eventuell während des gesamten Programms haben sollen. Viele Eigenschaften können Sie auch während der Laufzeit des Programms durch den Programmcode verändern lassen.

Typ	Eigenschaft	Einstellung
Formular	Text	MeinErstes
	Size	300; 300
Button	Name	`CmdHallo`
	Text	Hallo
Button	Name	`CmdEnde`
	Text	Ende
Label	Name	`LblAnzeige`
	Text	(leer)
	BorderStyle	`FixedSingle`

Tabelle 1.1 Steuerelemente mit Eigenschaften

Bei einem Label ergibt die Einstellung der Eigenschaft *BorderStyle* auf `FixedSingle` einen Rahmen. Zur Änderung auf `FixedSingle` müssen Sie die Liste bei der Eigenschaft aufklappen und den betreffenden Eintrag auswählen (siehe Abbildung 1.11). Zur Änderung einiger Eigenschaften müssen Sie eventuell sogar ein Dialogfeld aufrufen.

Im Label soll zunächst der Text *(leer)* erscheinen. Hierzu müssen Sie den vorhandenen Text durch Anklicken auswählen und ändern.

Liste der Steuerelemente

Sie finden alle in diesem 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 direkt untereinander.

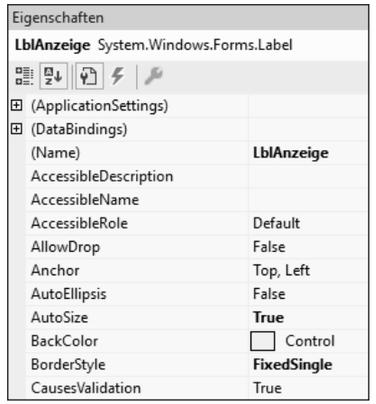

Abbildung 1.11 Label, Eigenschaften »Name« und »BorderStyle« geändert

1.3.4 Speichern eines Projekts

Alles speichern

Die Daten eines Visual Basic-Projekts werden in verschiedenen Dateien gespeichert. Zum Speichern des gesamten Projekts wird der Menüpunkt DATEI • ALLES SPEICHERN verwendet. Diesen Vorgang sollten Sie in regelmäßigen Abständen durchführen, damit keine Änderungen verloren gehen können.

Die in diesem Skript angegebenen Namen dienen als Empfehlung. Sie erleichtern die eindeutige Orientierung und das spätere Auffinden von alten Programmen.

1.3.5 Das Codefenster

Ereignis

Der Ablauf eines Windows-Programms wird im Wesentlichen durch das Auslösen von Ereignissen durch den Benutzer gesteuert. Er löst z. B. die Anzeige des Texts *Hallo* aus, indem er auf den Button HALLO klickt. Sie als Entwickler müssen dafür sorgen, dass aufgrund dieses Ereignisses der gewünschte Text angezeigt wird. Zu diesem Zweck schreiben Sie Programmcode und ordnen diesen Code dem Ereignis zu. Der Code wird in einer Ereignisprozedur abgelegt.

Ereignisprozedur

Zum Schreiben einer Ereignisprozedur führen Sie am besten einen Doppelklick auf das betreffende Steuerelement aus. Es erscheint das Codefenster. Zwischen der Formular- und der Code-Ansicht können Sie anschließend

über die Menüpunkte ANSICHT • CODE bzw. ANSICHT • DESIGNER hin- und herschalten. Das ist auch über die Registerkarten oberhalb des Formulars bzw. des Codefensters möglich (siehe Abbildung 1.12).

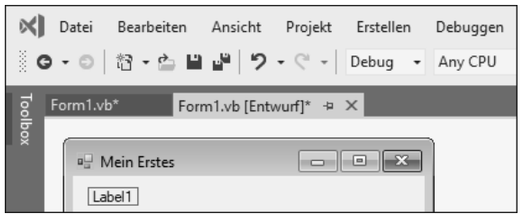

Abbildung 1.12 Registerkarten

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

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

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

Innerhalb der Ereignisprozedur ist der Platz für den eigenen Programmcode.

Class VB.NET ist eine objektorientierte Sprache. Ein wichtiges Element objektorientierter Sprachen sind Klassen. Alle Elemente des aktuellen Formulars `Form1` stehen innerhalb der Klasse `Form1` (zwischen `Public Class` und `End Class`). Auf die Einzelheiten der Objektorientierung wird zu einem späteren Zeitpunkt eingegangen, da das hier noch nicht notwendig ist und eher verwirren würde.

Sub Der Programmcode der Ereignisprozedur steht später zwischen `Private Sub` und `End Sub`. Der Name der Prozedur besteht aus den zwei Teilen *Name des Steuerelements* und *Ereignis*.

Zeilenumbruch Die Anweisung `Private Sub ...` ist recht lang, daher wird sie auf mehrere Zeilen verteilt. Visual Basic ermöglicht Zeilenumbrüche an bestimmten Stellen einer Anweisung auf einfache Art und Weise. Das erhöht wesentlich die Lesbarkeit von Programmen. Aus Druckgründen werden in diesem Buch häufig längere Anweisungen auf mehrere Zeilen verteilt. Die Regeln für Zeilenumbrüche finden Sie in Abschnitt 1.3.12. Der anfänglich ausgeführte Doppelklick führt immer zu dem Ereignis, das am häufigsten mit dem betreffenden Steuerelement verbunden wird. Das ist beim Button natürlich das Ereignis `Click`. Zu einem Steuerelement gibt es aber auch noch andere mögliche Ereignisse.

Oberhalb von `Public Class ...` und von `Private Sub ...` erscheinen seit Visual Studio 2015 automatisch weitere Informationen zur Klasse `Form1` und ihren Elementen in kleiner grauer Schrift, wie z. B. *0 Verweise* oder *2 Verweise* (engl.: *References*). Auch sie können wir zunächst ignorieren.

1.3.6 Schreiben von Programmcode

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

```
Public Class Form1
    Private Sub CmdHallo_Click(sender As Object,
            e As EventArgs) Handles CmdHallo.Click
        LblAnzeige.Text = "Hallo"
    End Sub
End Class
```

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

Der Text muss in Anführungszeichen gesetzt werden, da Visual Basic sonst annimmt, dass es sich um eine Variable mit dem Namen `Hallo` handelt.

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

Zuweisung Bei einer Zuweisung wird der Ausdruck rechts vom Gleichheitszeichen ausgewertet und der Variablen, der Objekt-Eigenschaft oder der Steuerelementeigenschaft links vom Gleichheitszeichen zugewiesen. Die Zeichenkette *Hallo* wird der Eigenschaft `Text` des Steuerelements `LblAnzeige` mithilfe 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 Ereignisprozedur geschrieben werden soll.

Code editieren Innerhalb des Codefensters können Sie Text mit den gängigen Methoden der Textverarbeitung editieren, kopieren, verschieben und löschen.

Syntaxfehler Sollten Sie bereits bei der Eingabe des Programmcodes Syntaxfehler gemacht haben, wird das angezeigt. Sie sollten den Code unmittelbar entsprechend korrigieren.

In der Ereignisprozedur `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 »MeinErstes«, Button »Ende«

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

Me Das aktuelle Objekt einer Klasse wird mit `Me` bezeichnet. Da wir uns innerhalb der Klasse für das aktuelle Formular befinden, bezieht sich `Me` auf dieses Formular. Die Anweisung zum Schließen dieses Formulars könnte also auch lauten: `Me.Close()`. In vielen Fällen ist `Me` allerdings überflüssig und kann weggelassen werden.

Graue Schriftfarbe Seit der Version 2015 von Visual Studio werden überflüssige Elemente im Programmcode deutlicher gekennzeichnet, indem sie in grauer Schriftfarbe dargestellt werden. Falls sich der Cursor in der betreffenden Zeile befindet, erscheint links von der Zeile eine gelbe Glühlampe. Platzieren Sie die Maus über dieser Glühlampe, wird der Zugang zu Hinweisen ermöglicht, wie die überflüssigen Elemente entfernt werden können. In Abbildung 1.13 sehen Sie ein Beispiel: Im Programmcode wurde `Me.Close()` statt `Close()` notiert.

Abbildung 1.13 »Me« kann entfernt werden.

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

1.3.7 Kommentare

Einfaches Hochkomma 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 verstehen. Eine Kommentarzeile beginnt mit einem einfachen Hochkomma. Alle Zeichen bis zum Ende der Zeile werden als Kommentar angesehen und folglich nicht übersetzt oder ausgeführt.

Der folgende Programmcode wird um eine Kommentarzeile ergänzt:

```
Private Sub CmdEnde_Click(sender As Object,
        e As EventArgs) Handles CmdEnde.Click
    ' Schließt die Anwendung
    Close()
End Sub
```

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

Code auskommentieren Ein kleiner Trick: Sollen bestimmte Programmzeilen für einen Test des Programms kurzfristig nicht ausgeführt werden, können Sie sie *auskommentieren*, indem Sie das Hochkomma vor die betreffenden Zeilen setzen. Das geht sehr schnell, indem Sie die betreffende(n) Zeile(n) markieren und anschließend das entsprechende Symbol im linken Bereich der Symbolleiste anklicken (siehe Abbildung 1.14). Rechts daneben befindet sich das Symbol, das die Auskommentierung nach dem Test wieder rückgängig macht.

Abbildung 1.14 Kommentar ein/aus

1.3.8 Starten, Ausführen und Beenden des Programms

Programm starten Nach dem Einfügen der Steuerelemente und dem Erstellen der Ereignisprozeduren ist das Programm fertig, und Sie können es starten. Dazu wird

der Start-Button in der Symbolleiste (dreieckiger grüner Pfeil nach rechts) betätigt. Alternativ starten Sie das Programm über die Funktionstaste F5 oder den Menüpunkt DEBUGGEN • DEBUGGING STARTEN. Das Formular erscheint, das Betätigen der Buttons führt zum programmierten Ergebnis.

Programm beenden

Zur regulären Beendigung eines Programms dient der Button mit der Aufschrift ENDE. Möchten Sie ein Programm während des Verlaufs abbrechen, können Sie auch den End-Button in der Symbolleiste (rotes Quadrat) betätigen.

Fehler

Tritt während der Ausführung eines Programms ein Fehler auf, wird das angezeigt, und das Codefenster zeigt die entsprechende Ereignisprozedur sowie die fehlerhafte Zeile an. Beenden Sie das Programm, korrigieren Sie den Code, und starten Sie das Programm wieder.

Programm testen

Es wird empfohlen, das Programm bereits während der Entwicklung mehrmals durch Aufruf zu testen und nicht erst, wenn das Programm vollständig erstellt worden ist. Geeignete Zeitpunkte sind z. B.

- nach dem Einfügen der Steuerelemente und dem Zuweisen der Eigenschaften, die sie zu Programmbeginn haben sollen,
- nach dem Erstellen jeder Ereignisprozedur.

1.3.9 Ausführbares Programm

.exe-Datei

Nach erfolgreichem Test des Programms könnten Sie auch die ausführbare Datei *(.exe*-Datei) außerhalb der Entwicklungsumgebung aufrufen. Sie finden sie im Unterverzeichnis *bin\Debug* des Projektverzeichnisses. Sie können das Programm also im Windows-Explorer direkt über einen Doppelklick starten.

Die Weitergabe eines eigenen Windows-Programms auf einen anderen PC ist etwas aufwendiger. Der Vorgang wird im Anhang beschrieben.

1.3.10 Projekt schließen, Projekt öffnen

Projekt schließen

Sie können ein Projekt über den Menüpunkt DATEI • PROJEKTMAPPE SCHLIESSEN schließen. Falls Sie Veränderungen vorgenommen haben, werden Sie gefragt, ob Sie diese Änderungen speichern möchten.

Möchten Sie die Projektdaten sicherheitshalber zwischendurch speichern, ist das über den Menüpunkt DATEI • ALLES SPEICHERN möglich. Das ist bei längeren Entwicklungsphasen sehr zu empfehlen.

Projekt öffnen

Zum Öffnen eines vorhandenen Projekts wählen Sie auf dem Startbildschirm links das betreffende Projekt direkt aus der Liste ZULETZT VERWENDETE ELEMENTE ÖFFNEN aus.

Sie können auch rechts auf dem Startbildschirm die Schaltfläche PROJEKT ODER PROJEKTMAPPE ÖFFNEN betätigen. Sollte der Startbildschirm nicht sichtbar sein, können Sie auch den Menüpunkt DATEI • ÖFFNEN • PROJEKT/PROJEKTMAPPE auswählen.

In beiden Fällen erscheint das Dialogfeld PROJEKT/PROJEKTMAPPE ÖFFNEN. Darin wählen Sie zunächst das gewünschte Projektverzeichnis aus und anschließend die gleichnamige Datei mit der Endung *.sln*.

Alle Beispielprojekte finden Sie auf *https://www.rheinwerk-verlag.de/4926* unter »Materialien zum Buch« zum Download. Sie können diese Projekte auf Ihre Festplatte kopieren. Sollte eines der Projekte einmal nicht gestartet werden können, sollten Sie es neu über den Menüpunkt ERSTELLEN • PROJEKTMAPPE NEU ERSTELLEN erstellen.

1.3.11 Übung

Übung »UName«

Erstellen Sie ein Windows-Programm mit einem Formular, das zwei Buttons und ein Label beinhaltet (siehe Abbildung 1.15). Bei Betätigung des ersten Buttons erscheint im Label Ihr Name. Bei Betätigung des zweiten Buttons wird das Programm beendet. Namensvorschläge: Projektname *UName*, Buttons *CmdMyName* und *CmdEnde*, Label *LblMyName*.

Abbildung 1.15 Übung »UName«

1.3.12 Regeln für Zeilenumbrüche

Zeilenumbruch

Visual Basic ermöglicht Zeilenumbrüche an bestimmten Stellen einer Anweisung auf einfache Art und Weise. Das erhöht die Lesbarkeit des Programmcodes. Einige Regeln wurden bereits im ersten Programm angewandt. Hier noch einmal die Prozedur `CmdHallo_Click()`:

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

Listing 1.5 Projekt »MeinErstes«, Ausschnitt

Regeln

Ein Zeilenumbruch ist möglich:

- nach einer öffnenden Klammer (siehe erste Prozedurzeile)
- vor einer schließenden Klammer (siehe dritte Prozedurzeile)
- nach einem Komma (siehe zweite Prozedurzeile)

Außerdem wäre ein Zeilenumbruch z. B. auch möglich:

- nach den meisten Operatoren, also auch nach dem Zuweisungsoperator `=` (siehe Abschnitt 2.2.5) oder dem Verkettungsoperator `&` (siehe Abschnitt 2.2.4).
- nach einem Punkt hinter einem Objektnamen, also auch nach dem Punkt hinter dem Objektnamen `LblAnzeige`

Auf keinen Fall dürfen Zeilenumbrüche innerhalb einer Zeichenkette durchgeführt werden. Weitere Möglichkeiten für Zeilenumbrüche werden jeweils an der passenden Stelle erläutert.

Unterstrich

Falls Sie das Zeichen _ (Unterstrich) am Ende einer Zeile notieren, kann anschließend in jedem Fall ein Zeilenumbruch durchgeführt werden, unabhängig von den oben genannten Regeln.

1.4 Arbeiten mit Steuerelementen

1.4.1 Steuerelemente formatieren

Hilfslinien

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

Mehrere Steuerelemente markieren

Weitere Möglichkeiten bieten die Menüpunkte im Menü Format. In vielen Fällen müssen vorher mehrere Steuerelemente auf einmal markiert werden (siehe Abbildung 1.16).

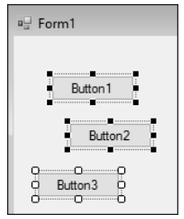

Abbildung 1.16 Mehrere markierte Elemente

Das geschieht entweder

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

Menü »Format«

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).
- Die horizontalen und/oder vertikalen Dimensionen der ausgewählten Steuerelemente können angeglichen werden (Menü Format • Grösse angleichen).
- **Einheitliche Abstände** Die horizontalen und vertikalen Abstände zwischen den ausgewählten Steuerelementen können 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- bzw. Hintergrund schieben (Menü Format • Reihenfolge).
- Sie können alle Steuerelemente gleichzeitig gegen versehentliches Verschieben absichern (Menü Format • Steuerelemente sperren). Diese Sperrung gilt nur während der Entwicklung des Programms.

Abbildung 1.17 zeigt ein Formular mit drei Buttons, die alle links ausgerichtet sind und im gleichen vertikalen Abstand zueinander stehen.

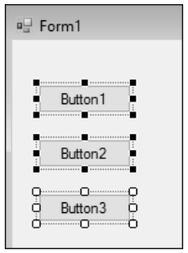

Abbildung 1.17 Nach der Formatierung

Übung

Laden Sie das Projekt *MeinErstes* aus dem vorherigen Abschnitt, markieren Sie darin mehrere Steuerelemente, und testen Sie die einzelnen Möglichkeiten des FORMAT-Menüs.

1.4.2 Steuerelemente kopieren

Steuerelemente kopieren

Zur schnelleren Erzeugung eines Projekts können vorhandene Steuerelemente einschließlich aller ihrer Eigenschaften kopiert werden. Markieren Sie hierzu die gewünschten Steuerelemente, und kopieren Sie sie entweder

- über das Menü BEARBEITEN • KOPIEREN und das Menü BEARBEITEN • EINFÜGEN oder
- mit den Tasten Strg + C und Strg + V.

Anschließend sollten die neu erzeugten Steuerelemente direkt umbenannt und an der gewünschten Stelle angeordnet werden.

Übung

Laden Sie das Projekt *MeinErstes* aus Abschnitt 1.3, und kopieren Sie einzelne Steuerelemente. Kontrollieren Sie anschließend die Liste der vorhandenen Steuerelemente im EIGENSCHAFTEN-Fenster auf einheitliche Namensgebung.

1.4.3 Eigenschaften zur Laufzeit ändern

Size, Location

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). Alle Werte werden in Pixeln gemessen.

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 Eigenschaft wie gewohnt im EIGENSCHAFTEN-Fenster eingeben. Als Beispiel für Änderungen während der Laufzeit soll das folgende Programm (Projekt *Steuerelemente*) dienen (siehe Abbildung 1.18).

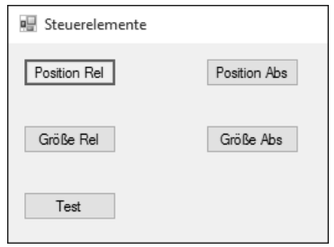

Abbildung 1.18 Position und Größe bestimmen

Der Programmcode:

```
Public Class Form1
    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, 200)
    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
End Class
```

Listing 1.6 Projekt »Steuerelemente«

Zur Erläuterung:

Verkürzte Darstellung

- Der Kopf der einzelnen Prozeduren wird aus Gründen der Übersichtlichkeit jeweils in verkürzter Form abgebildet. Das wird bei den meisten nachfolgenden Beispielen ebenfalls so sein, außer wenn es genau auf die Inhalte des Prozedurkopfs ankommt.
- Das Formular enthält 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` bzw. `Width`/`Height`).

New Point

- 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.

X, Y

- Bei Betätigung des Buttons Position Abs wird die Position des fünften Buttons auf die Werte `X=100` und `Y=200` gestellt, gemessen von der linken oberen Ecke des Formulars.
- Bei Betätigung des Buttons Position Rel 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 nicht nach oben oder unten.
- Zur Einstellung der Größe dient die Struktur `Size`.

Width, Height

- Bei Betätigung des Buttons Grösse Abs wird die Größe des fünften Buttons auf die Werte `Width = 50` und `Height = 100` gestellt.
- Bei Betätigung des Buttons Grösse Rel 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 alte Wert der Komponente `Height` nicht modifiziert, das Element verändert seine Höhe nicht.

Nach einigen Klicks sieht das Formular aus wie in Abbildung 1.19.

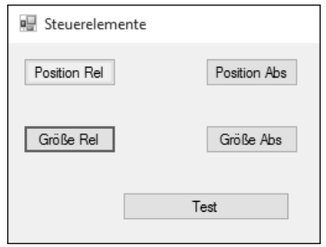

Abbildung 1.19 Veränderung von Eigenschaften zur Laufzeit

1.4.4 Vergabe und Verwendung von Namen

Beachten Sie in allen Programmen, dass jedes Steuerelement seinen eigenen, eindeutigen Namen hat und immer mit diesem Namen angesprochen werden muss. Es passiert erfahrungsgemäß besonders am Anfang häufig, dass ein Programm nicht zum gewünschten Erfolg führt, weil ein nicht vorhandener Name verwendet wird. In diesem Zusammenhang weise ich noch einmal auf die Namenskonventionen hin:

- Buttons sollten Namen wie `CmdEnde`, `CmdAnzeigen`, `CmdBerechnen` usw. haben.
- Labels sollten Namen wie z. B. `LblAnzeige`, `LblName`, `LblUhrzeit`, `LblBeginnDatum` haben.

Diese Namen liefern eine eindeutige Information über Typ und Funktion des Steuerelements. Falls Sie beim Schreiben von Programmcode anschließend diese Namen z. B. vollständig in Kleinbuchstaben eingeben, werden sie nach Verlassen der Zeile automatisch korrigiert. Daran können Sie schnell erkennen, ob Sie tatsächlich ein vorhandenes Steuerelement verwendet haben.

1.4.5 Verknüpfung von Texten, mehrzeilige Texte

& und vbCrLf

Es lassen sich mehrere Texte in einer Ausgabe mithilfe des Zeichens `&` miteinander verknüpfen. Falls Sie eine mehrzeilige Ausgabe wünschen, müs-

sen Sie zusätzlich einen Zeilenvorschub mithilfe der integrierten Visual Basic-Konstante `vbCrLf` eingeben.

Nachfolgend wird das Formular des Projekts *Steuerelemente* vergrößert, z. B. auf 500 × 400. Anschließend wird es um ein Label ergänzt, in dem die aktuelle Position und Größe des Buttons angezeigt wird. Das soll nach Betätigung des Buttons ANZEIGE geschehen:

```
Private Sub CmdAnzeige_Click(...) Handles ...
    LblAnzeige.Text =
        "Position: X: " & CmdTest.Location.X & ", Y: " &
        CmdTest.Location.Y & vbCrLf & "Größe: Breite: " &
        CmdTest.Size.Width & ", Höhe: " & CmdTest.Size.Height
End Sub
```

Listing 1.7 Projekt »Steuerelemente«, Anzeige

Nach einigen Klicks und der Betätigung des Buttons ANZEIGE sieht das Formular aus wie in Abbildung 1.20.

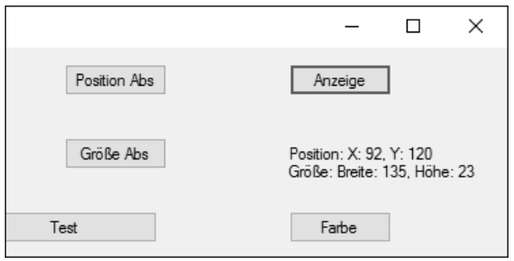

Abbildung 1.20 Anzeige der Eigenschaften

1.4.6 Eigenschaft BackColor, Farben allgemein

BackColor

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.

Color

Hintergrundfarben und andere Farben können auch zur Laufzeit eingestellt werden, dabei verwenden Sie Farbwerte. Diese Farbwerte können Sie über die Struktur `Color` auswählen.

Ein Beispiel, ebenfalls im Projekt *Steuerelemente*:

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

Listing 1.8 Projekt »Steuerelemente«, Anzeige der Farbe

Zur Erläuterung:

- Diese Struktur bietet vordefinierte Farbnamen als Eigenschaften, z. B. `Yellow`. Der Wert kann der Eigenschaft `BackColor` des Steuerelements zugewiesen werden, hier ist es das Formular selbst.
- Außerdem bietet die Struktur die Methode `FromArgb()`. Sie lässt sich auf verschiedene Arten aufrufen. Eine dieser Arten erwartet genau drei Parameter, nämlich die Werte für Rot, Grün und Blau, jeweils zwischen 0 und 255.

FromArgb()

Das Formular sieht nach der Änderung der Eigenschaft `Farbe` aus wie in Abbildung 1.21, wobei sich die Farbänderung bei der Schwarz-Weiß-Darstellung im gedruckten Buch leider nicht gut wiedergeben lässt.

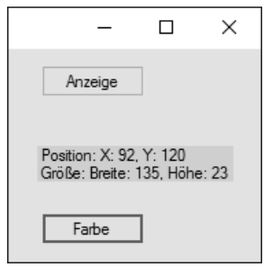

Abbildung 1.21 Nach Änderung der Eigenschaft »Farbe«

Kapitel 2
Grundlagen

In diesem Kapitel erlernen Sie auf anschauliche Weise die Sprachgrundlagen von Visual Basic 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 und häufig verwendeten Steuerelementen kennen.

2.1 Variablen und Datentypen

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

2.1.1 Namen und Werte

Namensregeln

Für die Namen von Variablen gelten in Visual Basic die folgenden Regeln:

- Sie beginnen mit einem Buchstaben.
- Sie können nur aus Buchstaben, Zahlen und einigen wenigen Sonderzeichen (z. B. dem Unterstrich) bestehen.

Sie dürfen Umlaute oder auch das scharfe S (auch Eszett genannt) enthalten. Allerdings kann das zu Fehlern im Einsatz unter anderssprachigen Umgebungen führen. Daher rate ich davon ab.

- Innerhalb eines Gültigkeitsbereichs darf es keine zwei Variablen mit dem gleichen Namen geben (siehe Abschnitt 2.1.4).

Variablen erhalten ihre Werte durch Zuweisung per Gleichheitszeichen. Falls eine Variable als Erstes auf der rechten Seite des Gleichheitszeichens

genutzt wird, sollte ihr vorher ein Wert zugewiesen werden. Dadurch werden Programme eindeutiger, lesbarer und fehlerfreier.

2.1.2 Deklarationen

Neben dem Namen besitzt jede Variable einen Datentyp, der die Art der Information bestimmt, die gespeichert werden kann. Sie als Entwickler wählen den Datentyp danach aus, ob Sie Text, Zahlen ohne Nachkommastellen, Zahlen mit Nachkommastellen oder z. B. logische Werte speichern möchten.

Auswahl des Datentyps

Außerdem müssen Sie sich noch Gedanken über die Größe des Bereichs machen, den die Zahl oder der Text annehmen könnte, und über die gewünschte Genauigkeit bei Zahlen. In Abschnitt 2.1.3 finden Sie eine Liste der Datentypen.

Variablen sollten in Visual Basic immer mit einem Datentyp deklariert werden. Das beugt Fehlern vor, die aufgrund einer falschen Verwendung der Variablen entstehen könnten.

2.1.3 Datentypen

Die wichtigsten von Visual Basic unterstützten Datentypen lassen sich in einige große Gruppen unterteilen:

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 **Integer**
- 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. $-3{,}4\times10^{38}$ bis ca. $3{,}4\times10^{38}$
- den Datentyp **`Double`** mit doppelter Genauigkeit und Werten von ca. $-1{,}7\times10^{308}$ bis ca. $1{,}7\times10^{308}$ **Double**
- den Datentyp **`Decimal`** mit variabler Genauigkeit und Werten von ca. $-7{,}9\times10^{28}$ bis ca. $7{,}9\times10^{28}$

Einige weitere nützliche Datentypen sind:

- der Datentyp **Boolean** mit den Werten `True` oder `False` (*wahr* oder *falsch*)

Date

- der Datentyp **Date** mit Werten für Datumsangaben vom 1. Januar des Jahres 1 bis zum 31. Dezember 9999
- der Datentyp **Char** für einzelne Zeichen

String

- 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 *Datentypen*).

```
Public Class Form1
    Private Sub CmdAnzeige_Click(...) Handles ...
        ' Ganze Zahlen
        Dim By As Byte
        Dim Sh As Short
        Dim It As Integer
        Dim Lg As Long

        ' Andere Zahlensysteme
        Dim Bn As Long
        Dim Hx As Long
        Dim Ot As Long

        ' Zahlen mit Nachkommastellen
        Dim Sg As Single
        Dim Db As Double
        Dim Dm 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 = 30_000
        It = 2_000_000_000
        Lg = 3_000_000_000

        ' Andere Zahlensysteme
        Bn = &B1100
        Hx = &H3F0
        Ot = &O37

        ' Zahlen mit Nachkommastellen
        Sg = 1 / 7
        Db = 1 / 7
        Dm = 1D / 7

        ' Boolesche Variable, Datum, Zeichen, Zeichenkette
        Bo = True
        Dt = "15.07.2019"
        Ch = "a"
        St = "Zeichenkette"

        LblAnzeige.Text =
           "Byte: " & By & vbCrLf &
           "Short: " & Sh & vbCrLf &
           "Integer: " & It & vbCrLf &
           "Long: " & Lg & vbCrLf & vbCrLf &
           "Binär: " & Bn & vbCrLf &
           "Hexadezimal: " & Hx & vbCrLf &
           "Oktal: " & Ot & vbCrLf & vbCrLf &
           "Single: " & Sg & vbCrLf &
           "Double: " & Db & vbCrLf &
           "Decimal: " & Dm & vbCrLf & vbCrLf &
           "Boolean: " & Bo & vbCrLf &
           "Date: " & Dt & vbCrLf &
           "Char: " & Ch & vbCrLf &
           "String: " & St
    End Sub
End Class
```

Listing 2.1 Projekt »Datentypen«

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

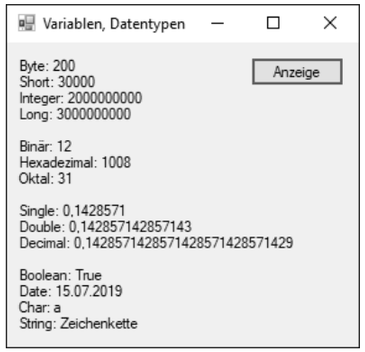

Abbildung 2.1 Variablen und Datentypen

Zur Erläuterung:

Dim ... As...

- Variablen werden mit `Dim ... As ...` deklariert. Mehrere Variablen desselben Datentyps können, durch Kommata getrennt, innerhalb einer Zeile deklariert werden (z. B. `Dim x, y As Integer`).
- Variablen können bereits bei der Deklaration einen Wert erhalten, z. B.: `Dim Sh As Short = 30000`.
- Bei den Datentypen für ganze Zahlen führt die Zuweisung einer zu großen Zahl zu einer Fehlermeldung.

Genauigkeit

- Die Datentypen für Zahlen mit Nachkommastellen unterscheiden sich in ihrer Genauigkeit. Nachkommastellen müssen im Programmcode durch einen Dezimalpunkt abgetrennt werden. In der Ausgabe erscheint dagegen ein Dezimalkomma. Die Zuweisung einer zu großen oder zu kleinen Zahl führt zur Anzeige von »Unendlich« bzw. zu einer ungenauen Speicherung.

Exponentialschreibweise

- Sehr große oder sehr kleine Zahlen lassen sich im Programmcode auch in der Exponentialschreibweise eingeben. Zwei Beispiele: 1.5e3 für 1500.0 oder 1.5e–3 für 0.0015. Beim Wechsel der Zeile während der Eingabe im Editor von Visual Studio werden diese Zahlen unmittelbar in Zahlen mit Nachkommastellen umgewandelt.

Zahlentrennzeichen _

- Zur Verbesserung der Lesbarkeit von Zahlen mit vielen Ziffern kann seit Visual Basic 2017 das Zahlentrennzeichen _ (Unterstrich) eingefügt werden, z. B. als Tausender-Trennzeichen.

Zahlensysteme

- Zur Eingabe von Zahlen können Sie seit Visual Basic 2017 auch das binäre, das hexadezimale oder das oktale Zahlensystem nutzen. Die

jeweilige Kennzeichnung der Zahlen und die Umrechnung in das normalerweise verwendete dezimale Zahlensystem werden am Ende des Abschnitts erläutert.

- Werte für Zeichen, Zeichenketten und Datumsvariablen müssen in doppelten Anführungszeichen angegeben werden.

Date

- Der Datentyp `Date` ist intern nur ein Alias, sprich: ein anderer Name für die Struktur `System.DateTime`. Mehr dazu in Abschnitt 6.2.
- Die Datentypen `Integer`, `Double`, `Boolean` und `String` werden am häufigsten eingesetzt.

Übung UDatentypen

Übung »UDatentypen«

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

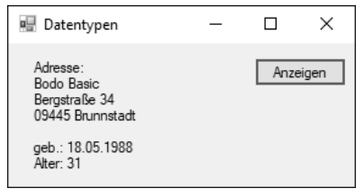

Abbildung 2.2 Übung UDatentypen

Hinweis

Das dezimale Zahlensystem hat bekanntlich die Basis 10. Falls Sie eine Zahl in einem anderen Zahlensystem eingeben möchten, müssen Sie die passende Kennzeichnung, das sogenannte Präfix, voranstellen. Zur Umrechnung der Werte aus den anderen Zahlensystemen:

- Beim binären Zahlensystem (Präfix: `&B`) dient die Zahl 2 als Basis. Die binäre Zahl `&B1100` wird wie folgt umgerechnet: $1\times2^3 + 1\times2^2 + 0\times2^1 + 0\times2^0 = 8 + 4 + 0 + 0 = 12$.
- Beim hexadezimalen Zahlensystem (Präfix: `&H`) wird die Zahl 16 als Basis verwendet. Es gibt weitere Ziffern: A=10, B=11, C=12, D=13, E=14 und F=15. Die hexadezimale Zahl `&H3F0` wird wie folgt umgerechnet: $3\times16^2 + 15\times16^1 + 0\times16^0 = 768 + 240 + 0 = 1008$.
- Das oktale Zahlensystem (Präfix: `&O`) hat die Basis 8. Die oktale Zahl `&O37` wird wie folgt umgerechnet: $3\times8^1 + 7\times8^0 = 24 + 7 = 31$.

2.1.4 Gültigkeitsbereich

Lokal

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

Static

Anders verhält es sich mit statischen Variablen. Sie behalten ihren Wert, solange das Programm läuft. Ein wiederholter Aufruf derselben Prozedur kann auf den letzten gespeicherten Wert einer Variablen zugreifen. Eine statische Variable vereinbaren Sie z. B. wie folgt: `Static Sx As Integer`.

Private, Dim

Variablen, die außerhalb von Prozeduren vereinbart werden, sind innerhalb der gesamten Klasse gültig, hier also innerhalb der Klasse des Formulars. Ihr Wert kann in jeder Prozedur gesetzt oder abgerufen werden und bleibt erhalten, solange das Formular im laufenden Programm existiert. Sie können die Variablen sowohl mit dem Schlüsselwort `Private` als auch mit dem Schlüsselwort `Dim` deklarieren. Sie sind außerhalb der Klasse nicht gültig. Weitere Einzelheiten zu klassenweit gültigen Variablen finden Sie in Kapitel 5.

Public

Variablen, die mit dem Schlüsselwort `Public` vereinbart werden, sind *öffentlich*. Damit sind sie auch außerhalb der jeweiligen Klasse gültig, also z. B. in anderen Formularen. Mehr dazu ebenfalls in Kapitel 5.

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

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

Ausblenden

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

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

```
Public Class Form1
    Private Mx As Integer

    Private Sub CmdAnzeigen1_Click(...) Handles ...
        Static Sx As Integer
        Dim x As Integer
        Sx += 1
        Mx += 1
        x += 1
        LblAnzeige.Text = "Sx:" & Sx & " x:" & x & " Mx:" & Mx
    End Sub

    Private Sub CmdAnzeigen2_Click(...) Handles ...
        Dim Mx As Integer
        Mx += 1
        LblAnzeige.Text = "Mx:" & Mx
    End Sub
End Class
```

Listing 2.2 Projekt »Gueltigkeitsbereich«

Zur Erläuterung:

- In der ersten Prozedur wird der Wert der statischen Variablen `Sx` und der klassenweit gültigen Variablen `Mx` bei jedem Aufruf um den Wert 1 erhöht. Die lokale Variable `x` wird dagegen jedes Mal von 0 auf 1 gesetzt (siehe Abbildung 2.3). Zur Erhöhung um den Wert 1 wird jeweils der kombinierte Zuweisungsoperator `+=` verwendet (siehe auch Abschnitt 2.2.5).

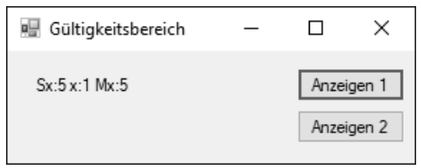

Abbildung 2.3 Lokale, statische und klassenweit gültige Variable

- In der zweiten Prozedur blendet die lokale Variable `Mx` die gleichnamige klassenweit gültige Variable aus. Die lokale Variable wird jedes Mal von 0 auf 1 gesetzt (siehe Abbildung 2.4).

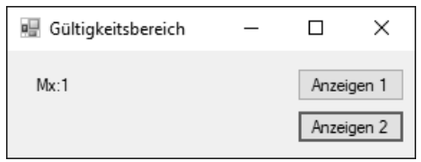

Abbildung 2.4 Lokale Variable

Hinweis

Die Variablen werden hier vor ihrer ersten Benutzung nicht initialisiert, d. h., sie werden nicht mit einem Startwert besetzt. In Visual Basic haben Zahlenvariablen zwar automatisch den Startwert 0, trotzdem sollten Sie die Initialisierung im Sinne eines sauberen Programmierstils im Normalfall selbst vornehmen.

Übung UGueltigkeitsbereich

Übung »UGueltigkeitsbereich«

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

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

In der ersten Prozedur werden `x` und `y` jeweils um 0,1 erhöht und angezeigt (siehe Abbildung 2.5). In der zweiten Prozedur werden `x` und `z` jeweils um 0,1 erhöht und angezeigt (siehe Abbildung 2.6).

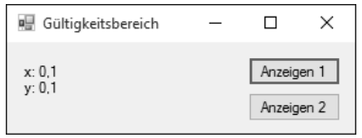

Abbildung 2.5 Ausgabe der ersten Methode nach einigen Klicks

Abbildung 2.6 Ausgabe der zweiten Methode nach weiteren Klicks

2.1.5 Konstanten

Konstanten repräsentieren Werte

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

Integrierte Konstanten

Zu den Konstanten zählen auch die integrierten Konstanten, wie z. B. `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 *Konstanten*).

```
Public Class Form1
    Private Const MaxWert = 75
    Private Const Eintrag = "Picture"

    Private Sub CmdKonstanten_Click(...) Handles ...
        Const MaxWert = 55
        Const MinWert = 5
        LblAnzeige.Text = (MaxWert - MinWert) / 2 &
            vbCrLf & Eintrag
    End Sub
End Class
```

Listing 2.3 Projekt »Konstanten«, Teil 1

Zur Erläuterung:

Const

- Konstanten werden mithilfe des Schlüsselworts `Const` definiert.
- Die Konstanten `MaxWert` und `Eintrag` werden mit klassenweiter Gültigkeit festgelegt.
- Innerhalb der Prozedur werden die beiden lokalen Konstanten `MaxWert` und `MinWert` festgelegt. `MaxWert` blendet die klassenweit gültige Konstante gleichen Namens aus, wie es in untenstehender Abbildung 2.7 zu sehen ist. IntelliSense weist bereits darauf hin, dass die klassenweit gültige Konstante nicht verwendet wird.
- Außerdem kommt noch die integrierte Konstante `vbCrLf` zum Einsatz.

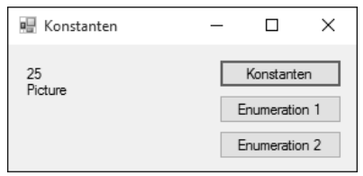

Abbildung 2.7 Konstanten

2.1.6 Enumerationen

Konstanten aufzählen

Enumerationen sind Aufzählungen von Konstanten, die thematisch zusammengehören. Alle Enumerationen haben den gleichen Datentyp, der ganzzahlig sein muss. Bei der Deklaration werden ihnen Werte zugewiesen, am besten explizit.

Innerhalb von Visual Basic gibt es 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 es Ihnen als Programmierer, die zahlreichen möglichen Antworten des Benutzers beim Einsatz von Windows-Standarddialogfeldern (JA, NEIN, ABBRECHEN, WIEDERHOLEN, IGNORIEREN ...) anschaulich einzusetzen.

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

```
Public Class Form1
[...]
    Private Enum Zahl As Integer
        Eins = 1
        Zwei = 2
        Drei = 3
        Vier = 4
    End Enum
[...]
    Private Sub CmdEnumeration1_Click(...) Handles ...
        LblAnzeige.Text = Zahl.Zwei * Zahl.Drei
    End Sub

    Private Sub CmdEnumeration2_Click(...) Handles ...
        LblAnzeige.Text = "Sonntag: " & FirstDayOfWeek.Sunday &
            ", Samstag: " & FirstDayOfWeek.Saturday
    End Sub
End Class
```

Listing 2.4 Projekt »Konstanten«, Teil 2

Zur Erläuterung:

Enum

- Enumerationen werden mithilfe des Schlüsselworts Enum definiert.
- Es wird die Enumeration Zahl vom Datentyp Integer vereinbart. Da es sich um einen Typ handelt und nicht um eine Variable oder Konstante, muss sie außerhalb von Prozeduren vereinbart werden. Damit ist sie automatisch für die gesamte Klasse gültig.
- In der ersten Ereignisprozedur werden zwei Elemente der eigenen Enumeration Zahl verwendet. Die beiden Zahlen, die sie repräsentieren, werden miteinander multipliziert (siehe Abbildung 2.8).

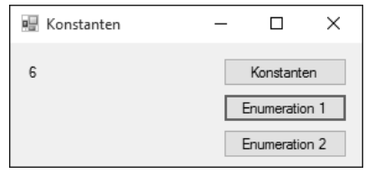

Abbildung 2.8 Erste Enumeration

- In der zweiten Ereignisprozedur werden zwei Elemente der vordefinierten Enumeration FirstDayOfWeek verwendet (siehe Abbildung 2.9). Sie können sie zur Ermittlung des Wochentags eines gegebenen Datums verwenden.

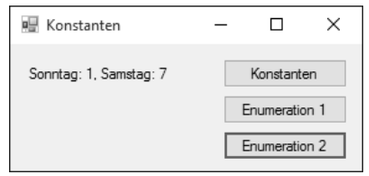

Abbildung 2.9 Zweite Enumeration

2.2 Operatoren

Zum Zusammensetzen von Ausdrücken nutzt Visual Basic, wie jede andere Programmiersprache auch, Operatoren. In diesem Buch wurden bereits die Operatoren = für Zuweisungen und & für Verkettungen verwendet.

Priorität

Es gibt verschiedene Kategorien von Operatoren. Vorrangregeln (Prioritäten) sind für die Reihenfolge der Abarbeitung zuständig, falls mehrere Operatoren innerhalb eines Ausdrucks eingesetzt werden. Diese Vorrangregeln sind weiter unten in diesem Abschnitt angegeben. Falls Sie sich bei der Ver-

wendung dieser Regeln nicht sicher sind, empfiehlt es sich, durch eigene Klammersetzung die Reihenfolge explizit festzulegen.

2.2.1 Rechenoperatoren

Rechenoperatoren

Die Rechenoperatoren dienen zur Durchführung von Berechnungen (siehe Tabelle 2.1).

Operator	Beschreibung
+	Addition
-	Subtraktion oder Negation
*	Multiplikation
/	Division
\	Ganzzahldivision
^	Potenzierung
Mod	Modulo

Tabelle 2.1 Rechenoperatoren

Ganzzahldivision

Die Ganzzahldivision 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.

Tabelle 2.2 zeigt einige Beispiele.

Ausdruck	Ergebnis
19 / 4	4.75
19 \ 4	4
19 \ 4.6	3
19.5 \ 4.2	5

Tabelle 2.2 Ganzzahldivision

Modulo

Der Modulo-Operator Mod berechnet den Rest einer Division. Zwei Beispiele sehen Sie in Tabelle 2.3.

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

Tabelle 2.3 Modulo-Operator

Zur Potenzierung einer Zahl dient der Operator ^ (hoch) (Beispiele siehe Tabelle 2.4).

Ausdruck	Ergebnis
2 ^ 5	32
3 ^ 2 ^ 3	729
2 ^ 5.4	42.2242531447326
-2 ^ 5	–32

Tabelle 2.4 Potenzierung

Multiplikation und Division innerhalb eines Ausdrucks sind gleichrangig und werden von links nach rechts in der Reihenfolge ihres Auftretens ausgewertet. Dasselbe gilt für Addition und Subtraktion, die zusammen in einem Ausdruck auftreten. Multiplikation und Division werden vor Addition und Subtraktion ausgeführt.

Klammern

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

Projekt

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

Übung URechenoperatoren

Übung »URechenoperatoren«

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

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

2.2.2 Vergleichsoperatoren

Vergleich

Vergleichsoperatoren (siehe Tabelle 2.5) dienen dazu, festzustellen, ob bestimmte Bedingungen zutreffen oder nicht. Das Ergebnis lässt sich u. a. zur Ablaufsteuerung von Programmen nutzen. Abschnitt 2.4 geht hierauf genauer ein.

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

Tabelle 2.5 Vergleichsoperatoren

Einige Beispiele sehen Sie in Tabelle 2.6.

Ausdruck	Ergebnis
`5 > 3`	`True`
`3 = 3.2`	`False`
`5 + 3 * 2 >= 12`	`False`

Tabelle 2.6 Nutzung von Vergleichsoperatoren

Like

Darüber hinaus gibt es noch den Operator `Like`, der zum Mustervergleich dient. Dabei können Sie u. a. die Platzhalter * (eines oder mehrere Zeichen) und ? (genau ein Zeichen) einsetzen. Tabelle 2.7 zeigt einige Beispiele.

Ausdruck	Ergebnis
`"abxba" Like "a*a"`	`True`
`"abxba" Like "a?a"`	`False`
`"aba" Like "a?a"`	`True`
`"asdlfigc" Like "a?d?f*c"`	`True`

Tabelle 2.7 Mustervergleich

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

Projekt

Übung UVergleichsoperatoren

Übung »UVergleichsoperatoren«

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

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

2.2.3 Logische Operatoren

Logik

Logische Operatoren dienen dazu, mehrere Bedingungen zusammenzufassen. Das Ergebnis lässt sich ebenfalls u. a. zur Ablaufsteuerung von Programmen nutzen (siehe hierzu auch Abschnitt 2.4). Die logischen Operatoren sehen Sie in Tabelle 2.8.

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 2.8 Logische Operatoren

Es seien die Variablen `A = 1`, `B = 3` und `C = 5` gesetzt. Die Ausdrücke in der ersten Spalte von Tabelle 2.9 führen zu den Ergebnissen 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 2.9 Ausdrücke mit logischen Operatoren

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

Projekt

Übung ULogische-Operatoren

Übung »ULogische-Operatoren«

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

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

2.2.4 Verkettungsoperator

Umwandlung in String

Der Operator `&` dient der Verkettung von Zeichenfolgen. Ist einer der Ausdrücke keine Zeichenfolge, sondern eine Zahl oder Datumsangabe, wird er in einen String umgewandelt. Das Gesamtergebnis ist wiederum eine Zeichenfolge. Beispiel:

```
Public Class Form1
    Private Sub CmdAnzeigen_Click(...) Handles ...
        Dim gesamt As String
        Dim s As Single
        Dim i As Integer
        Dim d As Date
        s = 4.6
        i = -12
        d = "10.08.2019"
        gesamt = "Hallo " & s & " " & i & " " & d
        LblAnzeige.Text = gesamt
    End Sub
End Class
```

Listing 2.5 Projekt »Verkettungsoperator«

Das Ergebnis ist in Abbildung 2.10 zu sehen.

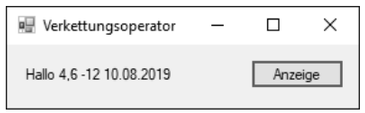

Abbildung 2.10 Verkettung

2.2.5 Zuweisungsoperatoren

Zeichen =

Der einfachste Zuweisungsoperator, das Gleichheitszeichen, wurde bereits genutzt. Es gibt zur Verkürzung von Anweisungen noch einige weitere

Zuweisungsoperatoren (siehe Tabelle 2.10). Zuweisungen mit diesen Operatoren werden auch Verbundzuweisungen genannt.

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`	Der Wert von `x` wird auf ein Drittel verringert. Nachkommastellen werden abgeschnitten.
`^=`	`x ^= 3`	Der Wert von `x` wird auf `x` hoch 3 erhöht.
`&=`	`z &= "abc"`	Die Zeichenkette `z` wird um den Text abc verlängert.

Tabelle 2.10 Zuweisungsoperatoren

2.2.6 Rangfolge der Operatoren

Priorität

Enthält ein Ausdruck mehrere Operationen, werden die einzelnen Teilausdrücke in einer bestimmten Rangfolge ausgewertet und aufgelöst, die als Rangfolge bzw. Priorität der Operatoren bezeichnet wird. Es gilt die Rangfolge in Tabelle 2.11.

Operator	Beschreibung
`^`	Exponentialoperator
`-`	negatives Vorzeichen
`*, /`	Multiplikation, Division
`\`	Ganzzahldivision
`Mod`	Modulo
`+, -`	Addition, Subtraktion
`&`	Verkettung

Tabelle 2.11 Rangfolge der Operatoren

Operator	Beschreibung
`=, <>, <, >, <=, >=, Like`	Vergleichsoperatoren (Das Zeichen `=` steht für den Vergleich, nicht für die Zuweisung.)
`Not`	logisches Nicht
`And`	logisches Und
`Or`	logisches Oder

Tabelle 2.11 Rangfolge der Operatoren (Forts.)

Die Operatoren, die in der Tabelle weiter oben stehen, haben die höchste Priorität.

Klammern

Wie bereits bei den Rechenoperatoren erwähnt: Mit Klammern können Sie diese Rangfolge außer Kraft setzen, damit bestimmte Teilausdrücke vor anderen Teilausdrücken ausgewertet werden.

In Klammern gesetzte Operationen haben grundsätzlich Vorrang. Innerhalb der Klammern gilt jedoch wieder die normale Rangfolge der Operatoren.

Übung UOperatoren

Übung »UOperatoren«

Sind die Bedingungen in Tabelle 2.12 wahr oder falsch? Lösen Sie die Aufgabe möglichst ohne PC.

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>=.9 And y<=5.8`
6	`x=1.0 y=5.7`	`x>=.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 2.12 Übung UOperatoren

2.3 Einfache Steuerelemente

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

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

2.3.1 Panel

Container

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

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

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

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

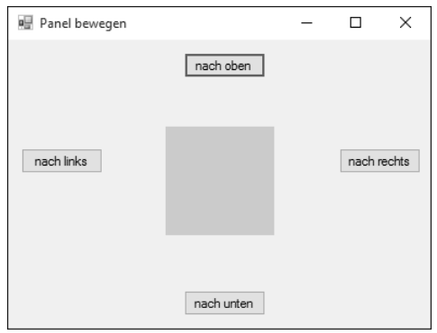

Abbildung 2.11 Panel, Startzustand

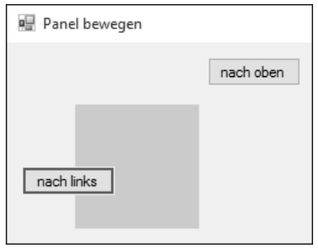

Abbildung 2.12 Panel, nach Verschiebung

Der Programmcode:

```
Public Class Form1
    Private Sub CmdOben_Click(...) Handles ...
        PanMove.Location = New Point(PanMove.Location.X,
            PanMove.Location.Y - 10)
    End Sub

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

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

    Private Sub CmdRechts_Click(...) Handles ...
        PanMove.Location = New Point(PanMove.Location.X + 10,
            PanMove.Location.Y)
    End Sub
End Class
```

Listing 2.6 Projekt »Panel«

2.3.2 Zeitgeber

Timer

Ein Zeitgeber (engl.: *Timer*) erzeugt in festgelegten Abständen Zeittakte. Diese Zeittakte sind Ereignisse, die Sie als Entwickler mit Aktionen verbinden können. Das zugehörige Ereignis heißt `Tick`. Ein Zeitgeber kann wie

jedes andere Steuerelement zum Formular hinzugefügt werden. Da es sich aber um ein nicht sichtbares Steuerelement handelt, wird er unterhalb des Formulars angezeigt. Auch zur Laufzeit ist er nicht sichtbar.

Intervall

Seine wichtigste Eigenschaft ist das Zeitintervall, in dem das Ereignis auftreten soll. Dieses Zeitintervall wird in Millisekunden angegeben.

Enabled

Die Eigenschaft `Enabled` dient der Aktivierung bzw. Deaktivierung des Zeitgebers. Sie kann zur Entwicklungszeit oder zur Laufzeit auf `True` oder `False` gestellt werden.

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

Abbildung 2.13 Nach einigen Sekunden

Der zugehörige Code:

```
Public Class Form1
    Private Sub TimAnzeige_Tick(...) Handles TimAnzeige.Tick
        LblAnzeige.Text &= "x"
    End Sub

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

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

Listing 2.7 Projekt »Zeitgeber«

Übung UPanelZeitgeber

Übung »UPanelZeitgeber«

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

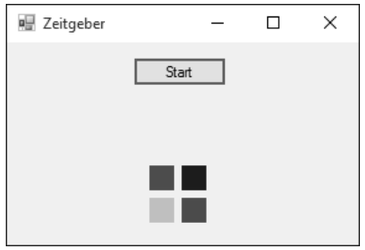

Abbildung 2.14 Startzustand

Sobald 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 2.15).

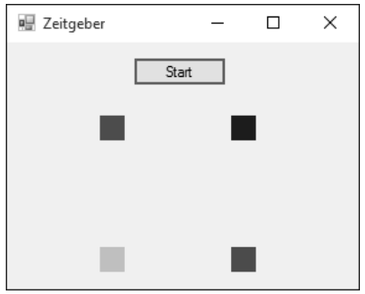

Abbildung 2.15 Nach einigen Sekunden

Übung UKran

Übung »UKran«

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

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

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

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

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

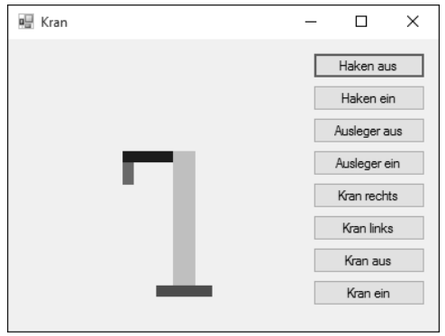

Abbildung 2.16 Startzustand

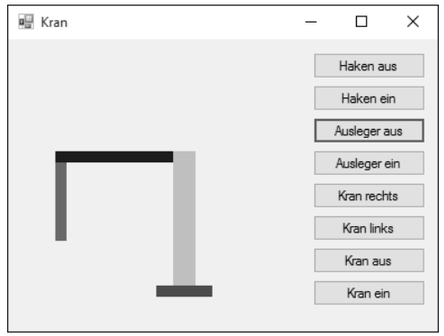

Abbildung 2.17 Nach einigen Aktionen

2.3.3 Textfelder

Eingabefeld

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

- `MultiLine`: Steht `MultiLine` auf `True`, kann bei der Eingabe und bei der Anzeige mit mehreren Textzeilen gearbeitet werden.
- `ScrollBars`: Ein Textfeld lässt sich mit vertikalen und/oder horizontalen Bildlaufleisten zur Eingabe und Anzeige längerer Texte versehen.
- `MaxLength`: Mit dieser Eigenschaft lässt sich die Anzahl der Zeichen des Textfelds beschränken. Ist keine Beschränkung vorgesehen, kann das Textfeld 32K Zeichen aufnehmen.

Passwort

- `PasswordChar`: Falls für diese Eigenschaft im Entwurfsmodus ein Platzhalterzeichen eingegeben wird, wird während der Laufzeit für jedes eingegebene Zeichen nur dieser Platzhalter angezeigt. Diese Eigenschaft wird vor allem bei Passwortabfragen verwendet.

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

Falls Sie umfangreichere Möglichkeiten zur Formatierung benötigen, z. B. zur Tief- oder Hochstellung einzelner Zeichen, empfehle ich das Steuerelement *RichTextBox* (siehe Abschnitt 7.8). Im nachfolgenden Programm im Projekt *Textfelder* kann der Benutzer in einem Textfeld einen Text eingeben. Nach Betätigung des Buttons AUSGABE wird der eingegebene Text in einem zusammenhängenden Satz ausgegeben (siehe Abbildung 2.18).

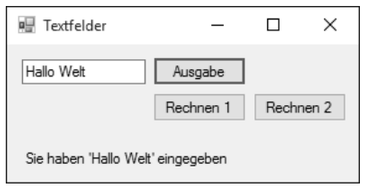

Abbildung 2.18 Eingabe in Textfeld

Der Code lautet wie folgt:

```
Public Class Form1
    Private Sub CmdAusgabe_Click(...) Handles ...
        LblAusgabe.Text = "Sie haben '" & TxtEingabe.Text &
            "' eingegeben"
    End Sub
End Class
```

Listing 2.8 Projekt »Textfelder«

Zur Erläuterung:

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

Zahlen eingeben

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

```
Private Sub CmdRechnen1_Click(...) Handles ...
    LblAusgabe.Text = TxtEingabe.Text * 2
End Sub
```

Listing 2.9 Projekt »Textfelder«, Zahleneingabe

Zur Erläuterung:

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

Einige Beispiele:

Abbildung 2.19 zeigt die Eingabe einer Zahl mit Nachkommastellen.

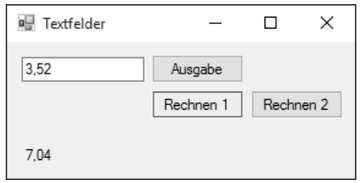

Abbildung 2.19 Eingabe einer Zahl mit Nachkommastellen

Debuggen beenden

Die Eingabe einer Zeichenkette, z. B. *abc*, führt zur Anzeige einer nicht behandelten Ausnahme. Die Zeile, in der der Fehler auftritt, wird im Code markiert, damit der Fehler beseitigt werden kann (siehe Abbildung 2.20). Das Programm muss über den Menüpunkt DEBUGGEN • DEBUGGEN BEENDEN beendet werden, bevor es sich neu starten lässt.

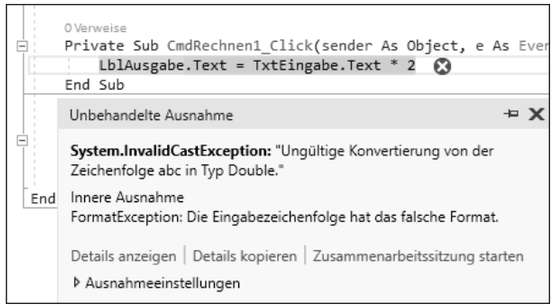

Abbildung 2.20 Anzeige des Fehlers

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

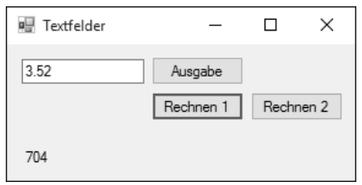

Abbildung 2.21 Punkt vor den Nachkommastellen

ToDouble()

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

```
Private Sub CmdRechnen2_Click(...) Handles ...
    LblAusgabe.Text = Convert.ToDouble(TxtEingabe.Text) * 2
End Sub
```

Listing 2.10 Projekt »Textfelder«, Methode ToDouble()

Zur Erläuterung:

- Der eingegebene Text wird mit der Methode `ToDouble()` in eine Zahl umgewandelt.
- Allerdings ist nach wie vor ein Programmabbruch bei falscher Eingabe möglich.

2.3.4 Zahlenauswahlfeld

NumericUpDown

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

Wichtige Eigenschaften des Steuerelements sind die folgenden:

Value

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

ValueChanged

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

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

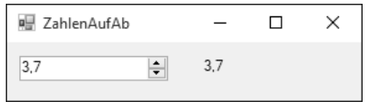

Abbildung 2.22 Zahlenauswahlfeld

Die Eigenschaften werden zur Entwicklungszeit wie folgt eingestellt:

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

Der Code lautet:

```
Public Class Form1
    Private Sub NumEingabe_ValueChanged(...) Handles _
            NumEingabe.ValueChanged
        LblAusgabe.Text = NumEingabe.Value
    End Sub
End Class
```

Listing 2.11 Projekt »ZahlenAufAb«

Anhang B
Lösungen der Übungsaufgaben

Die Bearbeitung der Beispiele, das Verständnis für ihren Aufbau sowie das selbstständige Lösen der Übungen sind ein wichtiger Schritt beim Erlernen der Programmierung. In diesem Abschnitt finden Sie jeweils eine Lösungsmöglichkeit zu jeder Übung. Lassen Sie sich nicht irritieren, wenn Ihre Lösung anders aussieht. Es ist in erster Linie wichtig, dass das Ergebnis stimmt. Vielleicht bietet Ihnen die hier vorliegende Lösung aber auch einen Denkanstoß zur Verbesserung.

B.1 Lösung der Übungsaufgabe aus Kapitel 1

B.1.1 Lösung UName

```
Public Class Form1
    Private Sub CmdMyName_Click(sender As Object,
            e As EventArgs) Handles CmdMyName.Click
        LblMyName.Text = "Bodo Basic"
    End Sub

    Private Sub CmdEnde_Click(sender As Object,
            e As EventArgs) Handles CmdEnde.Click
        Close()
    End Sub
End Class
```

B.2 Lösungen der Übungsaufgaben aus Kapitel 2

B.2.1 Lösung UDatentypen

```
Public Class Form1
    Private Sub CmdAnzeigen_Click(...) Handles ...
        Dim nachname, vorname As String
        Dim strasse, plz, ort As String
```



```
        Dim alter As Integer
        Dim geburtsdatum As Date

        nachname = "Basic"
        vorname = "Bodo"
        strasse = "Bergstraße 34"
        plz = "09445"
        ort = "Brunnstadt"
        alter = 31
        geburtsdatum = "18.05.1988"

        LblAnzeige.Text = "Adresse: " & vbCrLf & vorname & " " &
            nachname & vbCrLf & strasse & vbCrLf & plz & " " &
            ort & vbCrLf & vbCrLf & "geb.: " & geburtsdatum &
            vbCrLf & "Alter: " & alter
    End Sub
End Class
```

B.2.2 Lösung UGueltigkeitsbereich

```
Public Class Form1
    Private x As Double

    Private Sub CmdAnzeigen1_Click(...) Handles ...
        Dim y As Double
        y += 0.1
        x += 0.1
        LblAnzeige.Text = "x: " & x & vbCrLf & "y: " & y
    End Sub

    Private Sub CmdAnzeigen2_Click(...) Handles ...
        Dim z As Double
        z += 0.1
        x += 0.1
        LblAnzeige.Text = "x: " & x & vbCrLf & "z: " & z
    End Sub
End Class
```

B.2.3 Lösung URechenoperatoren

```
Public Class Form1
    Private Sub CmdAnzeigen1_Click(...) Handles ...
        Dim x As Double
        x = 3 * -2.5 + 4 * 2
        LblAnzeige.Text = x
    End Sub

    Private Sub CmdAnzeigen2_Click(...) Handles ...
        Dim x As Double
        x = 3 * (-2.5 + 4) * 2
        LblAnzeige.Text = x
    End Sub
End Class
```

Auf einen Blick

Inhalt