12 VBA Grundlagenwissen

Nachdem Sie nun bereits einige VBA-Programmiererfahrung gesammelt haben, möchte ich Ihnen in diesem und den nächsten Kapiteln das Handwerkszeug vorstellen, das Sie zum Programmieren brauchen. Keine Angst – auch hier gibt es wieder neben aller Theorie eine ganze Reihe praktischer Beispiele. Und denken Sie daran: »Anche i pesci del re hanno spine – Auch die Fische des Königs haben Gräten.« Also: Lassen Sie sich auch dieses Kapitel (trotz aller theoretischer Gräten) ein königliches Vergnügen sein!

12.1 Der Visual Basic-Editor

Das Fenster des *Visual Basic*-Editors haben Sie schon kennen gelernt. Dies ist die VBA-Entwicklungsumgebung, in der Sie Ihren VBA-Code entwickeln.

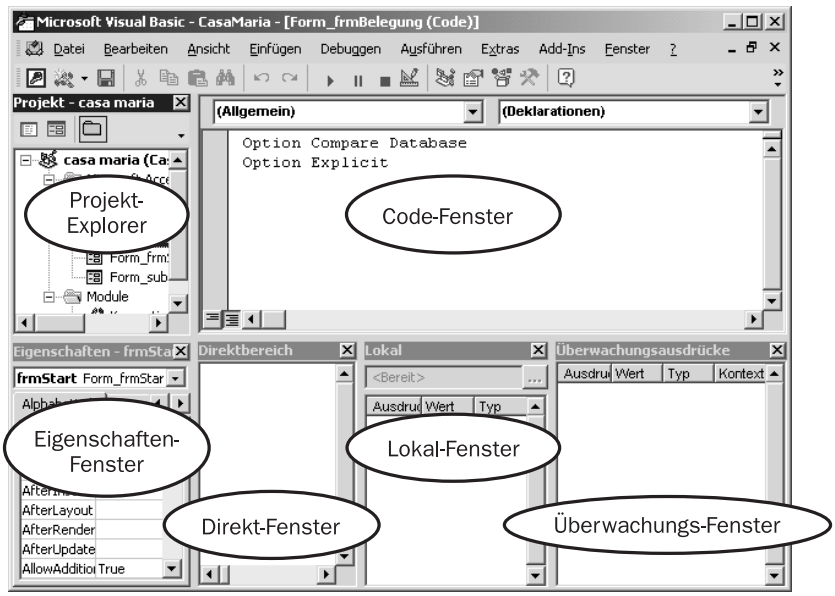


Bild 12.1: Die VBA-Entwicklungsumgebung mit allem Drum und Dran

Im Fenster des *Visual Basic*-Editors sind standardmäßig drei Unterfenster zu sehen:

Element	Beschreibung
Code-Fenster	Hier erstellen Sie Ihr Programm.
Projekt-Explorer	Hier sind alle Teile des aktuellen Programmierobjekts aufgelistet, also Module sowie Code für Formulare und Berichte.
Eigenschaften-Fenster	Hier werden die Eigenschaften des Objekts angezeigt, für das Sie gerade Code erfassen.

Tabelle 12.1: Standardfenster des *Visual Basic*-Editors

Über das Menü ANSICHT können bei Bedarf aufgerufen werden:

Element	Beschreibung
Direktfenster	Hier können Sie Befehle eingeben und direkt ausführen lassen, z.B. um das Programm zu testen.
Lokal-Fenster	Hier lassen sich Inhalte von Variablen einsehen.
Überwachungsfenster	Hier können Sie bestimmte Variablen überwachen.

Tabelle 12.2: Weitere Fenster über das Menü ANSICHT

Alle sechs Fenster lassen sich in Anordnung und Größe verändern oder schließen und über das Menü ANSICHT ein- und ausblenden. Zusätzlich lässt sich noch der Objektkatalog anzeigen, auf den ich in Kapitel 16 eingehe. Wie Sie im letzten Kapitel gemerkt haben, werden Sie zunächst vor allem mit dem Code-Fenster zu tun haben, in dem der Programmcode eingetragen wird.

Ansicht des Code-Fensters

Links unten im Code-Fenster finden Sie die beiden kleinen Schaltflächen mit den Bezeichnungen PROZEDURANSICHT und VOLLSTÄNDIGE MODULANSICHT, mit denen Sie zwischen den gleichnamigen Ansichten wechseln können.

In der Prozeduransicht wird im Code-Fenster jeweils nur eine einzelne Prozedur angezeigt. Mit der Tastenkombination Strg+↑ wechseln Sie zur vorigen bzw. mit Strg+↓ zur folgenden Prozedur.

12.2 Programmieren in Access

Im Prinzip ist ein Programm nichts anderes als eine Sammlung von Befehlen, die so angeordnet sind, dass sich mit ihnen eine bestimmte Aufgabe lösen lässt.

Beim Programmieren mit VBA wird die vom Programm zu erledigende Aufgabe in einzelne Teilaufgaben zerlegt. Zur Lösung einer Teilaufgabe werden einige Befehle zu einer so genannten Prozedur zusammengefasst. Im Programmablauf werden die einzelnen Teilaufgaben dann von den jeweiligen Prozeduren abgearbeitet. Im Zusammenspiel der einzelnen Prozeduren wird damit letztlich die Gesamtaufgabe gelöst.

12.2.1 Was wird programmiert?

Ähnlich wie bei Makros ist es auch möglich, VBA-Code entweder direkt an ein Formular- oder Berichtsereignis zu binden oder auch unabhängig von Formularen und Berichten aufzurufen.

Code behind Forms

Im letzten Kapitel haben Sie sich vor allem mit direkt einem Formular zugeordnetem Code beschäftigt (man spricht hier auch von code behind forms). Wie Sie vielleicht schon gemerkt haben, können Sie diesen VBA-Code nicht direkt über das *Access*-Datenbankfenster aufrufen. Hier besteht also ein Unterschied zu Makros, denn diese konnten auch dann direkt im Datenbankfenster gestartet werden, wenn sie an ein Formular gebunden waren.

Eigenständige Module

Lediglich das im letzten Kapitel vom Makro konvertierte Modul ist als eigenständiges Modul im Datenbankfenster abgelegt.

Zum Erstellen eines neuen eigenständigen Moduls

- ☑ rufen Sie einfach den Menübefehl EINFÜGEN ➧ MODUL auf, woraufhin sich der VBA-Editor öffnet.
- ☑ Rufen Sie im VBA-Editor den Menübefehl EINFÜGEN ➧ PROZEDUR auf und geben Sie der Prozedur den Namen `ToscanaTest`.

Bild 12.2:
Die TOSCANATEST-Prozedur

☑ Nachdem Sie mit OK bestätigt haben, tragen Sie in die Prozedur `Msgbox ("Ich bin ein eigenständiges Modul")` ein.

☑ Speichern Sie das Modul unter der vorgeschlagenen Bezeichnung `Modul1`.

Wenn Sie nun im Datenbankfenster auf MODUL1 klicken, wird nicht der Programmcode ausgeführt, sondern die Prozedur wird im VBA-Editor angezeigt. Um die Prozedur ausführen zu lassen,

☑ schreiben Sie `ToscanaTest` in das Direktfenster und drücken danach die ↵-Taste.

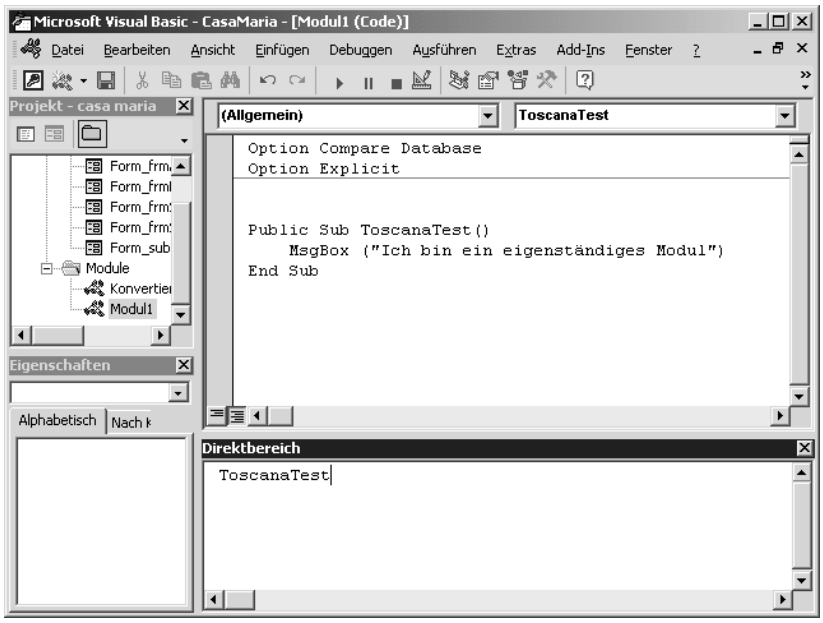


Bild 12.3:
Prozedur über Direktfenster testen

Jetzt erscheint die von Ihnen in der TOSCANATEST-Prozedur definierte Messagebox auf dem Bildschirm. Noch einfacher testen Sie eine Prozedur, indem Sie den Mauszeiger irgendwo innerhalb der Prozedur positionieren und dann die F5-Taste drücken oder in der Symbolleiste auf das Symbol SUB/USER FORM AUSFÜHREN klicken.

12.2.2 Wie wird programmiert?

Eine Prozedur wird mit der Zeile `Sub name()` begonnen und endet mit `End Sub`. `Sub`, `End`, `Private` usw. sind spezielle Ausdrücke von *Visual Basic*, so genannte Schlüsselwörter. Sie werden automatisch groß geschrieben und am Bildschirm blau dargestellt.

Schlüsselwörter grundsätzlich in Kleinbuchstaben eingeben!

Verlassen Sie dann eine Zeile, sollten alle Schlüsselwörter dieser Zeile automatisch in die richtige Groß-/Kleinschreibung umgesetzt werden. Geschieht dies nicht, ist das ein deutliches Zeichen dafür, dass Sie sich bei einem Schlüsselwort verschrieben haben.

Bei Prozeduren, die Sie für ein Steuerelement erstellen, wird der Name automatisch vergeben. Er besteht aus dem Namen des Steuerelements und des Ereignisses, das ausgeführt werden soll, um die Prozedur zu aktivieren, wie `Private Sub cmdSchliessen_Click()` für die Ereignisprozedur für einen Mausklick auf die Schaltfläche `cmdSchliessen`.

Für Prozeduren, die nicht direkt mit einem Steuerelement verbunden sind, können Sie den Namen der Prozedur im Prinzip frei wählen. Wird ein Prozedurname aus mehreren Wörtern zusammengesetzt, sollten Sie den Wortbeginn jeweils mit einem großen Buchstaben kennzeichnen, wie `ToscanaTest`.

Zwischen den beiden Zeilen, die den Beginn und das Ende einer Prozedur kennzeichnen, finden Sie die eigentlichen Anweisungszeilen. Dabei kann ein Programm im Prinzip aus einer einzigen Zeile oder aus hunderten von Zeilen bestehen.

Einrückung der Anweisungszeilen

Um ein Programm möglichst übersichtlich zu schreiben, rückt man zusammengehörende Zeilen ein. So wird beispielsweise alles, was innerhalb der Zeilen `Sub name()` und `End Sub` steht, eingerückt. Im folgenden Bild sehen Sie im Beispiel aus dem letzten Kapitel, dass auch die Anweisungszeilen, die innerhalb einer `If...End If`-Struktur stehen, eingerückt sind. Manchmal sagt man, dass Strukturen wie `Sub...End Sub`, `If...End If`, aber auch Schleifen, wie Sie sie noch kennen lernen werden, den dazwischen liegenden Programmcode »klammern«.

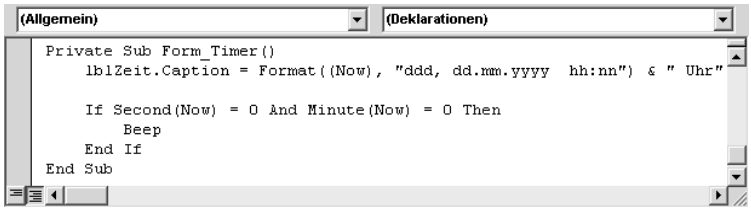


Bild 12.4: Eingerückte Anweisungszeilen

Kommentarzeilen

Damit Sie auch in ein paar Wochen noch verstehen können, was Sie einmal programmiert haben, ist es sinnvoll, so genannte Kommentarzeilen zwischen den Anweisungszeilen einzufügen. Manche Programmierer sagen sogar, ein gutes Programm enthält ebenso viele Kommentar- wie Anweisungszeilen.

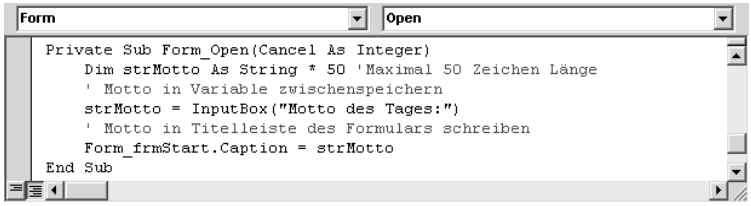


Bild 12.5: Kommentiertes Programm

Beginnen Sie die Kommentarzeile mit einem Hochkomma »'«. Sobald Sie die Zeile verlassen, wird sie grün dargestellt. Kurzkommentare können Sie auch am Ende einer Anweisungszeile schreiben, da alles nach einem Hochkomma als Kommentar angesehen wird.

Zeilenfortführung

Wird eine Zeile so lang, dass sie nicht mehr vernünftig im Code-Fenster angezeigt wird, können Sie solche Anweisungen auf zwei oder noch mehr Zeilen aufteilen. Wichtig ist dann nur, dass Sie jede Zeile, die fortgesetzt werden soll, mit einem Leerzeichen und einem Unterstrich »_« beenden.

12.3 Variablen besser verstehen

Im letzten Kapitel hatten Sie schon eine Variable definiert. Dabei hatten Sie die Variable als praktischen Zwischenspeicher kennen gelernt. Allgemein werden Variablen in Programmen eingesetzt, um Werte, Zeichenketten und anderes ablegen zu können, damit Sie später im Programm wieder darauf zugreifen können.

Variablen erhalten innerhalb Ihres Programms einen Namen, über den Sie auf den Inhalt der Variablen zugreifen können. Inhalt von Variablen können Zahlen, aber auch Datumsangaben, Uhrzeiten, Texte oder Bilder sein.

Variablen haben eine begrenzte Lebensdauer. In der Regel behalten sie ihren Wert höchstens so lange, bis die Prozedur beendet ist.

12.3.1 Variablendeklaration

Man sollte die Variablen, die in einem Programm verwendet werden, zu Beginn des Programms deklarieren und ihren Datentyp festlegen. Mit dem Vorgang des Deklarierens reserviert man sozusagen für die Variable Speicherplatz. Das ist zwar nicht unbedingt notwendig, hilft aber, Fehler zu verhindern.

Zwei unterschiedliche Arten der Variablendeklaration sind möglich, die explizite und die implizite Deklaration. Eine explizite Deklaration erfolgt dadurch, dass Sie zu Beginn des Programms eine Variable mit `Dim` deklarieren. Bei einer impliziten Deklaration erfolgt die Deklaration in dem Augenblick, in dem der Variablenname das erste Mal im Programm auftaucht, ohne dass Sie etwas dazu beitragen müssen.

Das klingt erst einmal nach Arbeitserleichterung, ist es aber unter Umständen gar nicht. Das Problem dabei ist nämlich, dass jede Schreibvariante einer Variablen automatisch als neue Variable angenommen wird. Falls Sie sich also versehentlich vertippen, haben Sie automatisch zwei verschiedene Variablen deklariert mit fast identischen Namen.

Daher ist es empfehlenswert, auf dem Registerblatt EDITOR (EXTRAS ➧ OPTIONEN) das Kontrollkästchen VARIABLENDEKLARATION ERFORDERLICH anzuklicken. Das zwingt Sie in neuen Projekten zur expliziten Variablendeklaration. Sie können das daran erkennen, dass der Code für ein neues Projekt mit der Zeile `Option Explicit` beginnt (siehe z.B. Bild 12.3).

Werden im Programm Variablen ohne vorherige Deklaration verwendet, wird der Variablenname nicht erkannt und Sie erhalten sofort eine Fehlermeldung. Dadurch lassen sich sehr leicht Tippfehler vermeiden.

Variablen richtig benennen

Variablen sollen so benannt werden, dass ihr Name möglichst kurz und prägnant sowie leicht zu behalten ist. Zudem muss ein Variablenname mit einem Buchstaben beginnen, darf maximal 256 Zeichen lang sein und außer dem »_« keine Sonderzeichen enthalten. Wird ein Variablenname aus mehreren Wörtern zusammengesetzt, so sollte jedes Wort gemäß der schon vorgestellten Namenskonvention groß geschrieben werden. Der Name lässt sich so leichter lesen, wie `EuroBetrag` oder `AppartementAnzahl`.

Verwenden Sie keine *Visual Basic*-Schlüsselwörter!

Achten Sie strikt darauf, dass Sie keine Schlüsselwörter, Objekt- oder Eigenschaftennamen als Namen für Ihre Variable verwenden.

12.3.2 Datentypen für Variablen

Außer den Speicherplatz durch die Deklaration zu reservieren, ist es auch sinnvoll, von vornherein festzulegen, um was für einen Datentyp es sich bei der Variablen handelt – ist es ein Text, eine Zahl, ein Datums- oder ein Wahrheitswert? Soll beispielsweise ein Text – wie »Franzis« – im Programm in einer Variablen abgelegt werden, so können Sie diese Variable durch

```
Dim Verlag As String
```

deklarieren. Mit dem Zusatz `As String` wird für die Variable der Datentyp »String«, Zeichenkette, festgelegt. Sie können dann mit

```
Verlag = "Franzis"
```

den Text in den Anführungszeichen der Variablen zuweisen.

Die folgende Tabelle zeigt die wichtigsten Datentypen für Variablen in *Access*.

Tabelle 12.3: Datentypen in *Visual Basic*

Datentyp	Art	Wertebereich
Byte	Ganze Zahlen	0 ... 255
Integer	Ganze Zahlen	-32.768 ... 32.767
Long	Ganze Zahlen	-2.147.483.648 ... 2.147.483.647
Single	Dezimalzahlen	Zahlen mit insgesamt 8 Stellen
Double	Dezimalzahlen	Zahlen mit insgesamt 16 Stellen
Currency	Dezimalzahlen für Währung	15 Vor- und 4 Dezimalstellen
Boolean	Wahrheitswerte	TRUE oder FALSE
Date	Datums-Zeit-Werte	1.1.100 bis 31.12.9999
String	Texte	kann 2 Milliarden Zeichen enthalten
Variant	kann beliebigen Datentyp enthalten (s.u.)	

Die Datentypen, die Sie in der Tabelle finden, sind nicht alle gleich wichtig. So verwendet man für ganze Zahlen, also Zahlen ohne Nachkommastellen, in der Regel den Datentyp `Integer`. Nur dann, wenn ganz sicher ist, dass die Zahlen immer kleiner als 255 sind,

würde man den Datentyp `Byte` verwenden, da dieser Datentyp sehr viel weniger Speicherplatz benötigt. Und nur dann, wenn Zahlen größer als 32.000 sind, wird der Datentyp `Long` verwendet. Möchten Sie Zahlen mit Nachkommastellen verwenden, werden diese Zahlen normalerweise als `Double` deklariert.

Geben Sie nicht explizit den gewünschten Datentyp an, wird eine Variable als `Variant` gespeichert. Eine Variable vom Datentyp `Variant` kann alles sein, ein Text, eine Zahl, ein Datum etc. Dieser Datentyp hat aber den Nachteil, dass er sehr viel mehr Speicherplatz benötigt als beispielsweise eine als Datum oder als Zahl deklarierte Variable. Zudem hilft auch das Festlegen des korrekten Datentyps Fehler zu vermeiden. Wird nämlich einer Variablen, die beispielsweise als Zahl festgelegt wurde, ein Text zugewiesen, erhalten Sie eine Fehlermeldung.

Dim-Anweisungen platzsparend

Bei Bedarf können Sie auch mehrere Dim-Anweisungen durch Komma getrennt in eine Zeile schreiben, wie

```
Dim strName As String, strKurs As String, dblBeitrag As
Double
```

Dabei ist allerdings darauf zu achten, dass man nicht versucht, zu viel Schreibarbeit zu sparen. Die Anweisung

```
Dim strName, strKurs As String, dblBeitrag As Double
```

nämlich würde `strName` als `Variant` deklarieren.

Zudem ist es sinnvoll, sich bei Variablennamen an die Namenskonventionen zu halten und den Datentyp einer Variablen durch eine Buchstabenkombination zu Beginn zu kennzeichnen. Die zu entsprechenden Abkürzungen finden Sie in folgender Tabelle.

Datentyp	Kürzel	Beispiel
Byte	byt	`Dim bytStatus As Byte`
Integer	int	`Dim intZähler As Integer`
Long	lng	`Dim lngEinwohner As Long`
Single	sng	`Dim sngKurs As Single`

Tabelle 12.4: Kürzel für Variablen entsprechend der Namenskonvention

Tabelle 12.4 (Forts.): Kürzel für Variablen entsprechend der Namenskonvention

Datentyp	Kürzel	Beispiel
Double	dbl	`Dim dblBetrag As Double`
Currency	cur	`Dim curDMBetrag As Currency`
Boolean	f	`Dim fMitglied As Boolean`
Date	dat	`Dim datGeburtsdatum As Date`
String	str	`Dim strName As String`
Variant	var	`Dim varDiverses As Variant`

Beispiel: Euro-Umrechnung mit Variablen

Im folgenden Beispiel geht es um die Euro-Umrechnung. Dazu wurde das Formular *frmEuroUmrechnung* erstellt. Sowohl die Zahleneingabe als auch die Ergebnisanzeige erfolgen im Textfeld TXTBETRAG, dem das Format STANDARDZAHL MIT ZWEI DEZIMALSTELLEN zugewiesen wurde. Die Optionsgruppe FRAEURODM gibt an, ob von DM in Euro oder umgekehrt gerechnet werden soll (das Präfix FRA der Optionsgruppe steht für frame, also Rahmen).

Bild 12.6: Euro-Umrechnung

Das folgende Programm wurde an das BEIM KLICKEN-Ereignis der Befehlsschaltfläche CMDUMRECHNEN gebunden. Die Umrechnung wird mithilfe der beiden Variablen `dblBetrag` und `dblKurs` ausgeführt.

```
Private Sub cmdUmrechnen_Click()
    Dim dblKurs As Double, dblBetrag As Double

    If IsNull(txtBetrag) Then
        txtBetrag = 0
    End If
```

```
    dblBetrag = txtBetrag
    dblKurs = 1.95583

    If fraEuroDM = 1 Then
        dblBetrag = dblBetrag / dblKurs
    Else
        dblBetrag = dblBetrag * dblKurs
    End If

    txtBetrag = Format(dblBetrag, "#,##0.00")
End Sub
```

Im Programm wurden die Variablen als `Double` deklariert. Das ist nicht unbedingt nötig gewesen, gerade der Kurswert mit seinen sechs Stellen hätte genauso gut als `Single` vereinbart werden können. Aber in der Regel ist der Speicherplatz heutzutage nicht mehr so knapp, dass man bei der Deklaration das Letzte herausholen muss.

Bevor den beiden Variablen die Werte übergeben werden, wird festgelegt, dass das Textfeld den Wert 0 annimmt, wenn nichts eingetragen ist. Dies ist erforderlich, damit der Variablen DBLBETRAG nicht der Wert Null zugeordnet werden kann, was einen Fehler verursachen würde.

Der Variablen `dblKurs` wird dann der Umrechnungskurs von `1,95583` (in der amerikanischen Schreibweise mit ».« anstelle dem Dezimalkomma) zugewiesen. Der Variablen `dblBetrag` wird der Inhalt des Textfeldes zugeordnet. Je nach der Eingabe in das Textfeld ist der Wert der Variablen `dblBetrag` unterschiedlich groß.

Danach gibt es für die Umrechnung zwei Möglichkeiten: Entweder ist für die Umrechnung von DM in Euro das Optionsfeld OPTDMEURO angeklickt, dann nimmt die Optionsgruppe FRAEURODM den Wert 1 an, oder es soll von Euro in DM umgerechnet werden, das Optionsfeld OPTEURODM ist angeklickt und FRAEURODM hat den Wert 2.

Zuletzt erfolgt hinter der `If...End If`-Konstruktion die Formatierung des errechneten Wertes als Zahl mit zwei Nachkommastellen und einem Tausenderpunkt (auch hier die Formatanweisung wieder in amerikanischer Schreibweise).

Mathematik gegen Informatik

Ist Ihnen in obigem Beispiel die Zeile

```
dblBetrag = dblBetrag * dblKurs
```

aufgefallen? Hier steht links und rechts des Gleichheitszeichens `dblBetrag`. Betrachten wir diese Formel mathematisch, so kann die Gleichung nach `dblKurs` aufgelöst werden und `dblKurs = 1`. Das ist natürlich Blödsinn, denn wir betrachten das Ganze mit den Augen der Informatik. Hier wird die Zeile wie folgt gelesen: Nimm den Wert, der im Speicher des Computers an der Stelle mit dem Namen `dblBetrag` steht, multipliziere ihn mit `dblKurs` und schreibe das Ergebnis wieder an den Speicherplatz `dblBetrag`.

Formatieren von Zahlen

Die `Format`-Funktion, die bereits mehrfach verwendet wurde, regelt die Darstellung von Zahlen. Die Formatierungsanweisung für Zahlen wird durch einen optionalen (#) und einen Mussplatzhalter (0) geregelt. Möchten Sie beispielsweise vier Nachkommastellen erzwingen, können Sie dies mit der Formatanweisung "0.0000" tun. Soll eine Zahl ohne Nachkommastellen, aber mit Tausenderpunkt formatiert werden, verwenden Sie "#,##0". Hierbei werden optionale Platzhalter verwendet, weil mit diesem Format nur Zahlen versehen werden sollen, die auch wirklich größer als 1.000 sind.

Bei der Formatierung von Zahlen sind zwei Dinge zu beachten: Zum einen ist immer die amerikanische Schreibweise (also mit vertauschtem Punkt und Komma) zu verwenden (was man leicht vergisst), zum anderen wird durch die `Format`-Funktion aus der Zahl eine Zeichenkette, also ein `String`.

Der Datentyp Boolean

Wahrheitswerte (`False` und `True`) werden mithilfe von Variablen des Typs `Boolean` verwaltet. In den folgenden Fragmenten einer

Prozedur wurde das Ergebnis der Checkbox (einem Kontrollkästchen) einer Variablen vom Typ `Boolean` zugewiesen.

Im Programm wird die Variable `fDMEuro` deklariert, der dann der Inhalt des Kontrollkästchens `chkDMEuro` zugewiesen wird. Dabei wird der Wert `1`, den das Kontrollkästchen zurückgibt, wenn es aktiviert wurde, automatisch in `True` umgewandelt. Der Wert `0` wird zu `False`. Daher lautet die Abfrage `If fDMEuro = True Then`.

```
Private Sub cmdUmrechnen_Click()
    ...
    Dim fDMEuro As Boolean

    ...
    fDMEuro = chkDMEuro

    If fDMEuro = True Then
        ...
    Else
        ...
    End If
    ...
End Sub
```

Wahrheitswerte in `If`-Abfragen

Verwenden Sie einen Wahrheitswert in einer `If`-Abfrage, können Sie auch kürzer `If fDMEuro Then` schreiben. Diese Bedingung ist dann erfüllt, wenn `fDMEuro` `True` ist.

Angenommen, Sie möchten dem Wahrheitswert `fFall` den Wert `True` dann zuordnen, wenn eine bestimmte Bedingung, wie `intAuswahl=4`, erfüllt ist. So könnten Sie

```
If intAuswahl = 4 Then fFall = True
```

schreiben oder kürzer

```
fFall=(intAuswahl=4)
```

Dabei wird zunächst die Klammer ausgewertet. Ist `intAuswahl=4`, wird `fFall True` zugeordnet; ergibt die Auswertung der Klammer, dass die Gleichung nicht erfüllt ist, wird `fFall` auf den Wert `False` gesetzt.

Der Datentyp Date

Datumswerte können das Zeitintervall vom 1. Januar 100 bis zum 31. Dezember 9999 abdecken. Uhrzeiten werden von 00:00:00 bis 23:59:59 dargestellt.

Möchten Sie der Variablen `datEintrittsdatum` vom Typ `Date` ein bestimmtes Datum zuweisen, so muss dieses Datum in »#« eingeschlossen werden. Sie können dabei ein Datum nicht so eingeben, wie Sie es gewohnt sind. Bei der Eingabe eines Datumswertes in Ihrem Programm ist eine der folgenden Schreibweisen zu verwenden:

```
datEintrittsdatum = #1 2 02#
datEintrittsdatum = #1 February 02#
datEintrittsdatum = #1 Feb 02#
datEintrittsdatum = #1,2,02#
```

Alternativ können Sie auch vierstellige Jahreszahlen verwenden. Sowie Sie diese Zeile verlassen, wird die Zeile automatisch in die amerikanische Schreibweise umgewandelt:

```
datEintrittsdatum = #2/1/02#   bzw.
datEintrittsdatum = #2/1/2002#
```

Achten Sie dabei darauf, dass die Reihenfolge des Tages und des Monats vertauscht wird. In der amerikanischen Schreibweise wird erst der Monat, dann der Tag genannt.

Die Anzeige eines Datums

erfolgt standardmäßig in der in der Ländereinstellung gewählten kurzen Datumsform. Möchten Sie eine andere Formatierung für Ihr Datum verwenden, benutzen Sie die `Format`-Funktion, so wie Sie es im letzten Kapitel für die Formatierung des im Formular angezeigten Datums gemacht haben. Dabei sind dann wieder die amerikanischen Abkürzungen zu verwenden:

yyyy	für Jahr
q	für Quartal (1 bis 4)
m	für Monat (1 bis 12)
d	für Monatstag (1 bis 31)
y	für Kalendertag (1 bis 366)
w	für Wochentag (1 bis 7)
ww	für Kalenderwoche (1 bis 53)

Setzen Sie so beispielsweise `"dddd, dd.mmmm yyyy"` für ein langes Datum in der Form `Mittwoch, 08.August 2001` zusammen.

Möchten Sie Uhrzeiten verwenden, sind diese ebenso in »#« einzuschließen. Dabei kann allerdings die gewohnte Schreibweise wie `datZeit = #7:00#` oder `datZeit = #19:00#` verwendet werden. Die Uhrzeiten werden dann in `datZeit = #7:00:00 AM#` bzw. `datZeit = #7:00:00 PM#` umgesetzt. Die Ausgabe entspricht dem in der Ländereinstellung ausgewählten Uhrzeitformat.

Anzeige der Uhrzeiten

Standardmäßig wird zur Darstellung von Uhrzeiten die in der Ländereinstellung dargestellte Form »`hh:mm:ss`« verwendet. Um eine Uhrzeit anders zu formatieren, müssen Sie in der `Format`-Funktion die Zeichen `h` für Stunde, `n` für Minute und `s` für Sekunde verwenden. Möchten Sie also eine Zeit ohne Sekunden angeben, verwenden Sie `"hh:nn"`. (Verwenden Sie lieber `n` zur Formatierung von Minuten, um Verwechslungen mit der Formatierung von Monaten zu vermeiden.)

Der Datentyp String

Variablen, die Texte enthalten sollen, sollten mit dem Datentyp `String` deklariert werden. Zur Übergabe eines Textes an eine Variable wird der Text in »"« eingeschlossen.

Leerer String

Ein leerer String – eine Zeichenkette, die kein Zeichen enthält – wird einfach durch »""« dargestellt.

Im folgenden Beispiel wird beim Öffnen des Formulars *frmString-Verarbeitung* der Inhalt zweier Strings, die einen Nachnamen und einen Vornamen enthalten, zu einem Gesamtnamen zusammengesetzt. Schließlich werden die Variablen an Textfelder übergeben und die entsprechenden Namen darin dargestellt.

```
Private Sub Form_Load()
    Dim strVorname As String
    Dim strNachname As String
    Dim strZusammengesetzterName As String

    strVorname = "Carla Louise"
    strNachname = "Klein"
    strZusammengesetzterName = strNachname & ", " & _
                               strVorname

    txtVorname = strVorname
    txtNachname = strNachname
    txtZusammengesetzterName = strZusammengesetzterName
End Sub
```

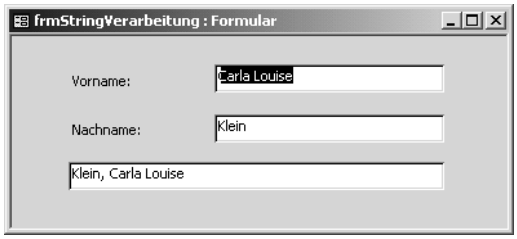


Bild 12.7: Beim Öffnen des Formulars wurde zusammengefügt

12.3.3 Konstanten

Im Prinzip sind Konstanten ein Sonderfall von Variablen. Konstanten wie

```
Const conPi = 3.14159265
Const conEuro = 1.95583
Const conAnzahl As Integer = 20
```

ändern sich während der Laufzeit eines Programms nicht. Konstanten erhalten zur Kennzeichnung die Vorsilbe »`con`«.

Warum sollten Sie Konstanten verwenden? Stellen Sie sich vor, Sie verwenden in Ihrem Programm an 20 verschiedenen Stellen die Zahl 11 für die Anzahl der zu Beginn am Euro teilnehmenden Nationen. Nun kommen im Laufe der Zeit neue Länder hinzu. Sie müssten jetzt an allen entsprechenden Stellen die 11 beispielsweise gegen eine 16 austauschen, wenn jetzt 16 Länder den Euro eingeführt hätten. Diese Ersetzung darf natürlich nur an den richtigen Positionen vorgenommen werden, Sie sollten keine 11 ersetzen, die gar nicht die Anzahl der Euro-Länder beschreibt.

Der Aufwand für eine solche Änderung lässt sich vereinfachen, indem Sie Konstanten verwenden. Definieren Sie

```
Const conEuroLänder = 11
```

und nutzen Sie diese Konstante anstelle der 11 in Ihrem Programm, so müssen Sie bei der Euro-Erweiterung auf 16 Länder Ihr Programm nur an einer Stelle ändern, nämlich nur die Definition der Konstante.

12.3.4 Felder

Ähnliche Werte werden üblicherweise in Feldern – auch Arrays genannt – zusammengefasst. Alle Werte eines Feldes können dann mit demselben Namen, aber über ihre Position im Feld – als Index bezeichnet – angesprochen werden.

Bei der Deklaration eines Feldes legt man gleich seine Größe fest. So steht beispielsweise die Zeile

```
Dim adblMwSt(2) As Double
```

für ein Feld mit drei Komponenten. Felder beginnen bei `0` zu zählen, damit sind für `adblMwSt` drei Elemente erlaubt, nämlich `adblMwSt(0)`, `adblMwSt(1)` und `adblMwSt(2)`.

Um im Variablennamen zu kennzeichnen, dass es sich um ein Feld handelt, wird dem Typkürzel ein »a« vorangestellt, das für die englische Bezeichnung `array` für ein Feld steht.

Die Zuweisung zu den einzelnen Werten eines Feldes erfolgt mithilfe der Indizes. Angenommen, Sie arbeiten mit den drei Mehrwertsteuersätzen 0%, 7% und 16%, dann schreiben Sie:

```
adblMwSt(0) = 0
adblMwSt(1) = 0.07
adblMwSt(2) = 0.16
```

Beispiel: MwSt-Berechnung mit Feldern

Im Formular *frmMwSt* habe ich die Optionsgruppe fraMwSt eingefügt, die Werte zwischen 0 und 2 annehmen kann, je nachdem, welches Optionsfeld angeklickt wurde. Daneben gibt es noch die Textfelder txtNetto für die Eingabe des Nettowertes und txtBrutto für die Ausgabe des berechneten Bruttowertes. Beiden habe ich das Format `Währung` gegeben. Ein Klick auf die Schaltfläche cmdBerechnen startet die Berechnung für den gewählten Mehrwertsteuersatz. Dazu habe ich die folgende Prozedur an das Beim Klicken-Ereignis der Schaltfläche gebunden:

```
Private Sub cmdBerechnen_Click()

    Dim dblBrutto As Double, dblNetto As Double
    Dim intIndex As Integer
    Dim adblMwSt(2) As Double

    ' MwSt-Werte zuordnen
    adblMwSt(0) = 0
    adblMwSt(1) = 0.07
    adblMwSt(2) = 0.16

    ' Sicherstellen, dass nicht Null übergeben wird
    If IsNull(txtNetto) Then
        txtNetto = 0
    End If

    intIndex = fraMwSt
```

```
    dblNetto = txtNetto
    dblBrutto = dblNetto * adblMwSt(intIndex) + dblNetto
    txtBrutto = dblBrutto

End Sub
```

Die Array-Funktion

Zum Füllen von Feldern können Sie auch die `Array`-Funktion verwenden. Der Vorteil besteht darin, dass die Werte einer `Array`-Funktion in Form einer Liste zugeordnet werden, was Schreibarbeit spart. Die Werte eine `Array`-Feldes sind immer vom Datentyp `Variant`.

Ein `Array`-Feld müssen Sie deklarieren, es braucht aber nicht dimensioniert werden. So wäre beispielsweise

```
Dim avarMwSt As Variant
```

ausreichend. Um das Feld mit Werten zu füllen, schreiben Sie

```
avarMwSt = Array(0, 0.07, 0.16)
```

Die einzelnen Werte der Liste werden dabei durch Kommata voneinander getrennt.

12.4 Gültigkeit von Variablen

In diesem Abschnitt soll die Frage »Wie lange lebt eine Variable?« geklärt werden.

12.4.1 Gültigkeit auf Prozedurebene

Werden Variablen innerhalb einer Prozedur deklariert, sind sie nur innerhalb dieser Prozedur bekannt; man sagt, sie gelten auf Prozedurebene. Man bezeichnet solche Variablen auch als lokale Variablen.

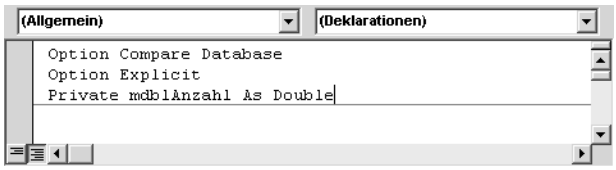
In einer anderen Prozedur ist die entsprechende Variable unbekannt, dort muss sie neu deklariert werden. Allerdings ist es dabei

nicht möglich, auf den Wert einer Variablen von einer anderen Prozedur her zuzugreifen.

12.4.2 Gültigkeit auf privater Modulebene

Benötigen Sie Variablen nicht nur in einer Prozedur, sondern müssen mehrere Prozeduren auf dieselbe Variable zugreifen, muss eine solche Variable auf Modulebene deklariert werden. Sie gilt dann in allen Prozeduren eines Formulars. Solche Variablen werden direkt unter der Zeile `Option Explicit` vor den eigentlichen Prozeduren mit dem Zusatz `Private` anstelle von `Dim` deklariert. (Es ist auch möglich, zur Deklaration einer Variablen auf privater Modulebene `Dim` zu verwenden, allerdings macht die Verwendung von `Private` die Gültigkeitsebene deutlicher und ist daher vorzuziehen.)

Bild 12.8:
Modulweite Deklaration



Zur Kennzeichnung der modulweiten Gültigkeit sollten Sie ein »m« vor das eigentliche Typkürzel stellen.

12.4.3 Gültigkeit auf öffentlicher Modulebene

Soll eine Variable nicht nur innerhalb eines Moduls gelten, sondern in einer gesamten Anwendung, wird sie auf öffentlicher Modulebene deklariert. Sie können sowohl aus Formularen oder Berichten als auch anderen Modulen darauf zugreifen. Öffentliche Variablen werden im Deklarationsbereich eines Moduls mit `Public` deklariert und erhalten zur Kennzeichnung den Buchstaben »g« für global. Sie werden auch als globale Variablen bezeichnet.

12.4.4 Variablen mit gleichem Namen

Was geschieht, wenn eine Variable mit gleichem Namen, die auf Prozedurebene – also lokal – in zwei unterschiedlichen Prozeduren deklariert wurde, verwendet wird? Das ist in diesem Fall kein Pro-

blem, da eine lokale Variable nur innerhalb der Prozedur, in der sie deklariert wurde, bekannt ist. Wird eine zweite Prozedur aufgerufen und eine Variable mit demselben Namen verwendet, so weiß diese nichts von der anderen Variablen. Entsprechend ist der Wert der Variablen – falls ihr in der zweiten Prozedur nicht explizit ein Wert zugewiesen wird – gleich Null.

Was geschieht, wenn eine Variable mit gleichem Namen, zum einen auf Modulebene, zum anderen lokal auf Prozedurebene deklariert wird? (Verwenden Sie für modulweite Variablen die Kennzeichnung »m«, kann das eigentlich gar nicht passieren.) In einem solchen Fall gilt innerhalb der Prozedur, in der die lokale Variable deklariert ist, nur diese. Die modulweit deklarierte Variable wird ignoriert. Das bedeutet für den Wert der lokalen Variablen, dass sie unabhängig vom Wert der modulweiten Variablen gleich Null ist, bis ihr in der Prozedur ein Wert zugewiesen wird.

12.5 Operatoren

Möchten Sie beim Programmieren Formeln verwenden, so müssen Sie natürlich auch die Operatoren kennen lernen, die in den Formeln eingesetzt werden können. Einige der Operatoren des folgenden Überblicks haben Sie bei Ihrer Arbeit mit *Access* schon kennen gelernt, wie »+«, »/« und »*«.

12.5.1 Welche Operatoren gibt es?

Grob kann man die Operatoren in arithmetische (oder mathematische), logische, Vergleichs- und Verkettungsoperatoren unterscheiden.

Arithmetische Operatoren

Die folgende Tabelle zeigt eine Zusammenfassung der arithmetischen Operatoren, die zu Berechnungen benötigt werden.

Tabelle 12.5: Arithmetische Operatoren

Operator	Operation
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
\	führt eine ganzzahlige Division durch. Dabei wird das Ergebnis nicht gerundet, sondern die Nachkommastellen einfach abgeschnitten, so dass beispielsweise `29\5` den Wert `5` ergibt.
^	Potenzierung
Mod	führt eine ganzzahlige Division durch. Dieser Operator gibt dann den ganzzahligen Rest zurück, so dass beispielsweise `29 Mod 5` den Wert `4` ergibt.

Vergleichsoperatoren

Mithilfe von Vergleichsoperatoren werden Werte oder Ausdrücke miteinander verglichen, wie `intAnzahl < 100`. Vergleichsoperatoren benötigen Sie in `If`-Abfragen oder `Select Case`-Anweisungen (siehe Abschnitt 12.6).

Tabelle 12.6: Vergleichsoperatoren

Operator	Operation
=	überprüft auf Gleichheit
<, >	kleiner als bzw. größer als
<=, >=	kleiner oder gleich bzw. größer oder gleich
<>	ungleich

Logische Operatoren

In Bedingungen können nicht nur Vergleichsoperatoren, sondern auch logische Operatoren verwendet werden, wie Sie sie in der folgenden Tabelle sehen.

Operator	Beispiel	Bedeutung
And	`IntAnzahl > 10 And intAnzahl < 90`	`intAnzahl` soll größer `10` und kleiner `90` sein, also zwischen `10` und `90` liegen.
Or	`IntAnzahl < 10 Or intAnzahl > 20`	`intAnzahl` soll entweder kleiner als `10` oder größer als `20` sein.
Not	`Not (intAnzahl < 10)`	`intAnzahl` soll nicht kleiner als `10` sein, also soll `intAnzahl` größer oder gleich `10` sein.

Tabelle 12.7: Logische Operatoren

Zwei mit `Or` verbundene Bedingungen sind dann erfüllt, wenn eine der beiden erfüllt ist. Im Unterschied dazu ist eine mit `And` verbundene Bedingung nur dann erfüllt, wenn beide Bedingungen wahr sind. Es besteht zudem die Möglichkeit, mehrere logische Operatoren zusammen in einer Bedingung zu verwenden, wie in

```
Not (intAnzahl < 10) And Not (intAnzahl > 90)
```

Da es nicht immer ganz einfach ist festzulegen, wie welcher Operator arbeitet, soll die folgende Tabelle mit einer Zusammenfassung des Zusammenspiels der einzelnen Operatoren helfen. Dabei bedeutet die Bedingung `True`, dass die angegebene Bedingung erfüllt ist. Ist das Ergebnis gleich `True`, heißt das, dass die Verkettung der beiden Bedingungen als wahr angegeben wird.

Bedingung1	Operator	Bedingung 2	Ergebnis
True	And	True	True
True	And	False	False
False	And	True	False
False	And	False	True
True	Or	True	True
True	Or	False	True
False	Or	True	True
False	Or	False	False
	Not	True	False
	Not	False	True

Tabelle 12.8: Zusammenspiel der logischen Operatoren

Verkettungsoperatoren

Es gibt zwei Verkettungsoperatoren: »+« und »&«. Mit dem »+«-Operator werden allgemein Ausdrücke und Werte miteinander verkettet, wie `intAnzahl + intZuwachs`. Es handelt sich hierbei um den Additionsoperator.

Der Operator »&« hingegen verkettet Zeichenfolgen, wie

```
strGanzerName = strVorname & strName
```

oder

```
strWohnort = "85586 " & "Poing"
```

12.5.2 Reihenfolge der Auswertung von Operatoren

Operatoren untereinander sind nicht gleichwertig. Beispielsweise wird im Ausdruck `dblTeilSumme1 + dblTeilSumme2 / dblKurs` zuerst die Division und erst dann die Summe ausgewertet.

Ganz allgemein werden zuerst die arithmetischen Operatoren bzw. die Verknüpfungsoperatoren für Zeichenketten ausgewertet, dann die Vergleichsoperatoren und zum Schluss die logischen.

Befinden sich in einem Ausdruck mehrere gleichwertige Operatoren, so werden sie von links nach rechts berücksichtigt.

Die folgende Tabelle zeigt die Reihenfolge der arithmetischen Operatoren untereinander.

Tabelle 12.9: Reihenfolge der arithmetischen Operatoren

^	Potenzierung
-	Negation
* bzw. /	Multiplikation bzw. Division
\	Ganzzahldivision
Mod	Restwert
+ bzw. -	Addition bzw. Subtraktion

Die Vergleichsoperatoren sind untereinander alle gleichwertig. Für die logischen Operatoren gilt die folgende Reihenfolge:

`Not`
`And`
`Or`

Tabelle 12.10: Reihenfolge der logischen Operatoren

Wünschen Sie eine andere Reihenfolge der Auswertung?

Dann verwenden Sie Klammern! Soll in obigem Beispiel zuerst die Summe der Teilsummen berechnet werden und dann die Division, können Sie `(dblTeilSumme1 + dblTeilSumme2) / dblKurs` schreiben.

12.6 Bedingte Abfragen und Verzweigungen

Mithilfe von Abfragen können Sie in Ihrem Programm erreichen, dass nur bestimmte Teile ausgeführt werden. Dazu lassen sich Bedingungen angeben, die regeln, welche Teile in einem Durchlauf ausgeführt werden und welche nicht. Hierzu die beiden wichtigsten Strukturen sind `If`-Abfragen und `Select Case`-Anweisungen.

12.6.1 If-Abfragen

Mithilfe einer `If`-Abfrage lassen Sie einen bestimmten Teil Ihres Programms nur dann ablaufen, wenn die angegebene Bedingung erfüllt ist. Die einfachste Form einer `If`-Abfrage lautet

`If` *`Bedingung`* `Then` *`Anweisung`*

Eine solche einzeilige `If`-Anweisung besteht aus den Schlüsselwörtern `If` und `Then`. Dazwischen wird eine Bedingung angegeben. Für die Bedingung stehen Ihnen die im vorherigen Abschnitt besprochenen Operatoren zur Verfügung. Ist die Bedingung erfüllt, wird die Anweisung ausgeführt, die hinter dem Schlüsselwort `Then` aufgeführt ist, sonst nicht.

Beispielsweise könnten Sie eine solche Bedingung dann verwenden, wenn

```
If intVerkaufteExemplare > 10000 Then dblRabatt = 0.03
```

Mehr Flexibilität erhält man mit mehrzeiligen, geschachtelten Abfragen:

```
If Bedingung Then
    [Anweisungen]
[ElseIf Bedingung 1 Then
    [SonstWennAnweisungen]]
[ElseIf Bedingung 2 Then
    [SonstWennAnweisungen]]
[ElseIf Bedingung 3 Then
    [SonstWennAnweisungen]]
...
[Else
    [SonstAnweisungen]]
End If
```

Bei dieser Schreibweise werden die Teile in eckige Klammern geschrieben, die nicht unbedingt notwendig, also optional sind. So kann eine solche Anweisung nur aus

```
If Bedingung Then
    Anweisungen
End If
```

aber auch nur aus

```
If Bedingung Then
    Anweisungen
Else
    SonstAnweisungen
End If
```

bestehen, da die `Else`- und `ElseIf`-Abschnitte beide optional sind. Sie können in einem `If`-Block beliebig viele `ElseIf`-Abschnitte

verwenden. Allerdings darf sich hinter einer `Else`-Zeile keine `ElseIf`-Anweisung mehr befinden.

Bei der Programmausführung wird ein `If`-Block von oben nach unten abgearbeitet. Zunächst wird die Bedingung hinter dem `If` überprüft. Ist sie erfüllt, ergibt sie also den Wert `True`, werden die nachfolgenden Anweisungen ausgeführt. Danach wird das Programm mit der Anweisung `End If` fortgeführt. Falls das nicht der Fall ist, werden nacheinander alle `ElseIf-Bedingungen` – sofern es welche gibt – geprüft. Ist keine der Bedingungen `True` gewesen, wird zum Schluss die `Else`-Anweisung ausgeführt.

Einrückungen

Bei `If`-Anweisungen ist es sehr hilfreich, mit Einrückungen zu arbeiten. So lässt sich beispielsweise ein vergessenes `End If` relativ schnell finden.

Bei `If`-Anweisungen besteht auch die Möglichkeit, mehrere `If`-Abfragen ineinander zu schachteln.

Beispiel: Rabatt bei langer Buchungsdauer

Je nachdem, wie lange die Casa-Maria-Appartements gebucht werden, soll ein unterschiedlich großer Rabatt angeboten werden. Ab 29 Tagen werden 8%, zwischen 15 und 28 Tagen 4% und zwischen 10 und 14 Tagen nur noch 2% Rabatt angeboten.

Dazu habe ich das Formular *frmRabattrechner* erstellt. Hier werden Anreise- und Abreisedatum eingetragen sowie der Tagespreis des Appartements. Nach Klick auf die Schaltfläche CMDBERECHNEN werden dann im unteren Teil der Gesamtbetrag, der Rabattbetrag und der ermäßigte Betrag ausgegeben.

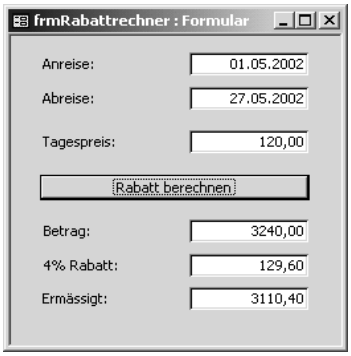


Bild 12.9:
Berechnen, was unterm Strich rauskommt

In der zugehörigen Prozedur gibt es eine Reihe von `If`-Abfragen. Die Gültigkeitsabfragen am Anfang werden jeweils mit einer Meldung und `Exit sub` beendet, wodurch der Programmablauf unterbrochen wird.

Sind die Angaben im oberen Teil des Formulars richtig gemacht, folgt im Programmablauf nach der Berechnung der Anzahl der Tage die eigentliche Definition der Rabattstaffel. Am Ende werden dann die berechneten Werte in die Textfelder TXTBETRAG, TXTRABATT bzw. TXTBETRAGERMÄSSIGT geschrieben. Außerdem wird die Höhe des Rabatts in das Bezeichnungsfeld LBLRABATT eingetragen.

```
Option Compare Database
Option Explicit

Private Sub cmbBerechnen_Click()
    Dim dblAnzahl As Double
    Dim dblRabatt As Double, dblBetrag As Double

    ' Keine Null-Werte zulassen - Einträge erzwingen
    If IsNull(txtTagespreis) Then
        MsgBox ("Bitte Tagespreis eintragen")
        Exit Sub ' Prozedur verlassen
    End If

    If IsNull(txtAbreise) Or IsNull(txtAnreise) Then
        MsgBox ("Bitte Anreise und Abreise eintragen")
        Exit Sub
    End If

    ' Anreise vor Abreise überprüfen,
    ' da dblAnzahl sonst negativ werden würde
    If txtAbreise < txtAnreise Then
        MsgBox ("Anreise muss vor Abreise liegen")
```

```
        Exit Sub
    End If

    ' Anzahl der Tage berechnen
    dblAnzahl = txtAbreise - txtAnreise + 1

    ' Rabattstaffel definieren
    If dblAnzahl >= 29 Then
        dblRabatt = 0.08
    ElseIf dblAnzahl >= 15 Then
        dblRabatt = 0.04
    ElseIf dblAnzahl >= 10 Then
        dblRabatt = 0.02
    End If

    ' Betrag ohne Rabatt
    dblBetrag = dblAnzahl * txtTagespreis
    txtBetrag = dblBetrag

    ' Rabatt in Bezeichnungsfeld schreiben
    lblRabatt.Caption = Format(dblRabatt, "#0% Rabatt:")
    ' Rabattbetrag
    dblBetrag = dblBetrag * dblRabatt
    txtRabatt = dblBetrag

    ' Ermäßigter Betrag
    dblBetrag = txtBetrag - txtRabatt
    ' Ermäßigten Betrag ausgeben
    txtBetragErmässigt = dblBetrag

End Sub
```

12.6.2 Select Case-Anweisungen

Verzweigungen sind außer mithilfe einer `If`-Struktur auch mit einer `Select Case`-Anweisung möglich. Gerade bei `If`-Strukturen mit mehreren `ElseIf`s ist eine `Select Case`-Struktur oft übersichtlicher.

Allgemein lässt sich eine `Select Case`-Anweisung durch

```
Select Case Variable
    [Case Ausdruck 1
        [Anweisungen 1]
```

```
    [Case Ausdruck 2
        [Anweisungen 2]
    ...
    [Case Ausdruck n
        [Anweisungen n]]]]
    [Case Else
        [SonstAnweisungen]]
End Select
```

ausdrücken. Eine solche Anweisung wird durch die Schlüsselwörter `Select Case` und `End Select` eingeschlossen. In der ersten Zeile wird eine Variable angegeben, deren Wert in den späteren `Case`-Zeilen überprüft wird. Ist der Ausdruck, der nach `Case` folgt, für die Variable erfüllt, werden die folgenden Anweisungen abgearbeitet. Ist der Ausdruck nicht erfüllt, wird der nächste `Case`-Ausdruck geprüft. Trifft keine der hinter `Case` ausgeführten Bedingungen zu, werden – falls vorhanden – die *SonstAnweisungen* nach `Case Else` ausgeführt.

Beispiel: Rabattstaffel mit Select Case

Die Rabattstaffel soll nun folgendermaßen erweitert werden:

Tabelle 12.11: Neue Rabattstaffel

Anzahl Tage	Rabatt
0 – 7	0%
8 – 14	2%
15 – 21	4%
22 – 28	6%
29 – 35	8%
ab 36	10%

Um die Programmierung mit `Select Case` zu testen, müssen Sie einfach nur den Rabattstaffel-`If`-Block des letzten Beispiels durch die folgende `Select Case`-Struktur ersetzen.

```
Select Case dblAnzahl
    Case Is >= 36
        dblRabatt = 0.1
    Case Is >= 29
```

```
        dblRabatt = 0.08
    Case Is >= 22
        dblRabatt = 0.06
    Case Is >= 15
        dblRabatt = 0.04
    Case Is >= 8
        dblRabatt = 0.02
End select
```

Prozentzahlen

Achten Sie darauf, wenn Sie mit Prozentzahlen rechnen möchten, eine solche Zahl durch 100 zu teilen, so dass 100% der Zahl 1 entsprechen, 4% hingegen der Zahl 0,04 (bzw. 0.04 in amerikanischer Schreibweise).

12.7 Wiederholte Anweisungen: Schleifen

Schleifen sind Programmieranweisungen, die es Ihnen erlauben, Teile Ihres Programms mehrfach hintereinander auszuführen. Die drei wichtigsten Möglichkeiten, Schleifen zu programmieren, sollen Ihnen in den folgenden Abschnitten vorgestellt werden.

12.7.1 For...Next

Die `For...Next`-Schleife verwendet einen Zähler, der mitzählt, wie häufig der in die Schleife eingeschlossene Programmteil bereits durchlaufen wurde. Ist die angegebene Anzahl von Durchläufen erreicht, wird die Prozedur hinter der Anweisung `Next` weitergeführt.

Allgemein kann man eine `For..Next`-Schleife durch

```
For Zähler = Anfang To Ende [Step Schrittweite]
    [Anweisungen]
    [Exit For]
    [Anweisungen]
Next [Zähler]
```

ausdrücken. Dabei ist `Zähler` eine Variable, die als ganze Zahl (beispielsweise als `Integer`) deklariert wurde. `Anfang` und `Ende` geben den Anfang- bzw. Endwert für den Zähler an, wie `For intZähler = 0 To 10`. Die folgenden Anweisungen würden elf Mal durchlaufen werden, nämlich für die Werte `intZähler = 0,1,..10`. Der in Klammern geschriebene Teil `Step Schrittweite` wird nur benötigt, wenn der Zähler nicht – was er standardmäßig tut – um eins weitergezählt werden soll, sondern beispielsweise um `2` oder um `-1`, wie in den folgenden beiden Zeilen:

```
For intGeradeZahlen = 2 To 20 Step 2
For intRückwärts = 10 To 0 Step -1
```

In der ersten Zeile nimmt der Zähler `intGeradeZahlen` die Werte `2,4,6...20` an, in der zweiten Zeile durchläuft `intRückwärts` die Werte `10,9,8...0`.

Die Anweisungen, die zwischen den Schlüsselwörtern `For` und `Next` liegen, werden nun solange durchlaufen, bis der Zähler den Wert `Ende` erreicht hat, es sei denn, es befindet sich eine Bedingung in den Anweisungen, die einen vorzeitigen Abbruch erzwingt. Ein solcher Abbruch erfolgt durch die Zeile `Exit For`. Die Prozedur wird dann bei der ersten Anweisung fortgesetzt, die der Zeile `Next` folgt.

Beispiel: Bericht mehrmals drucken mit For...Next

Nach Klick auf die Schaltfläche ADRESSETIKETTEN im Formular *frmStart* wird in diesem Beispiel in einer Inputbox abgefragt, wie oft der Bericht gedruckt werden soll. Diese Angabe wird dann in einer `For...Next`-Schleife verarbeitet. Mit der `IsNumeric`-Funktion wird dabei festgestellt, ob die Eingabe eine Zahl ist und mit `DoCmd.OpenReport` wird der Bericht gedruckt.

```
Private Sub cmdAdressetiketten_Click()
    Dim varAnzahl
    Dim intZähler As Integer

    varAnzahl = InputBox _
    ("Wie oft soll die Adressenliste gedruckt werden?")
    ' Abbrechen, nichts oder 0 eingeben führt zu Abbruch
    If varAnzahl = "" Then Exit Sub
    If varAnzahl < 1 Then Exit Sub
```

```
    If IsNumeric(varAnzahl) Then
        ' Ggf. vorhandene Kommastellen abschneiden
        varAnzahl = varAnzahl \ 1

        For intZähler = 1 To varAnzahl
            DoCmd.OpenReport _
               ("rptEtikettenAdressenDeutschland")
        Next
    Else
        ' Meldung, wenn keine Zahl eingegeben wurde
        MsgBox ("Bitte ganze Zahl eingeben!")
    End If

End Sub
```

12.7.2 Do...Loop

Für eine `Do...Loop`-Schleife gibt es verschiedene Varianten. Eine solche Schleife läuft solange (`While`) oder bis (`Until`) eine Bedingung erfüllt ist. Allgemein lässt sich eine `Do...Loop`-Schleife in zwei unterschiedlichen Formen schreiben, entweder die Bedingung wird gleich zu Anfang abgefragt, wie in

```
Do {While | Until} Bedingung
    [Anweisungen]
    [Exit Do]
    [Anweisungen]
Loop
```

oder die Bedingung wird nach der eigentlichen Schleife kontrolliert, wie in

```
Do
    [Anweisungen]
    [Exit Do]
    [Anweisungen]
Loop {While | Until} Bedingung
```

Die zweite Form gewährleistet, dass die `Do...Loop`-Schleife mindestens einmal ausgeführt wird, wohingegen die erste Form keinmal, einmal oder mehrmals ausgeführt werden kann.

Beispiel: Bericht mehrmals drucken mit Do...Loop

Um das letzte Beispiel mit `Do...Loop` zu lösen, müssen Sie den `For...Next`-Programmteil ersetzen durch eine der beiden folgenden `Do...Loop`-Lösungen. Da bei dieser Art von Schleife nicht automatisch weitergezählt wird, geschieht das hier mit der Zeile `intZähler = intZähler + 1`.

```
intZähler = 0
Do While intZähler <= varAnzahl
    DoCmd.OpenReport ("rptEtikettenAdressenDeutschland")
    intZähler = intZähler + 1
Loop
```

Im folgenden Ausschnitt aus einer Prozedur läuft die Schleife so lange, bis der Wert von `intZähler` größer ist als `varAnzahl`:

```
intZähler = 0
Do Until intZähler > varAnzahl
    DoCmd.OpenReport ("rptEtikettenAdressenDeutschland")
    intZähler = intZähler + 1
Loop
```

Bericht am besten ohne `Do...Loop` und ohne `For...Next` drucken

Beim Ausdrucken eines Berichtes in einer Schleife muss der Bericht jedes Mal neu generiert werden. Besser ist es, hierfür die Anzahl der Kopien anzugeben, wie in folgendem Programmfragment:

```
' Bericht öffnen
DoCmd.OpenReport "rptEtikettenAdressenDeutschland",
acViewPreview

' Ausdruck der gewählten Anzahl von Kopien
DoCmd.PrintOut PrintRange:=acPages, PageFrom:=1,
PageTo:=1, Copies:=varAnzahl

' Bericht schließen
DoCmd.Close ObjectType:=acReport,
ObjectName:="rptEtikettenAdressenDeutschland"
```

12.8 Springen mit GoTo

Manchmal ist es erforderlich, dass man innerhalb der Prozedur eine bestimmte Zeile anspringt und von dort die Prozedur fortsetzt. Dabei sind `GoTo`'s hilfreich. Zu viele `GoTo`'s sollten Sie innerhalb Ihrer Programme vermeiden, da sonst der Programmcode schwer lesbar wird. Um ein `GoTo` verwenden zu können, müssen Sie angeben, wohin gesprungen werden soll. Dazu verwendet man so genannte Sprungmarken. Das sind einfach Bezeichnungen vor derjenigen Zeile, die angesprungen werden soll. Eine solche Sprungmarke erhält zur Erkennung einen »:«.

Im folgenden Ausschnitt einer Prozedur soll dann zur Sprungmarke `Ende` gesprungen werden, wenn die Bedingung `intNummer > 10000` erfüllt ist.

```
...
If intNummer > 10000 GoTo Ende
...
Ende:
End Sub
```