

## Kapitel 3

# Das ABC der Programmierung

*In diesem Kapitel werden die Grundregeln der Programmierung erklärt, die nicht nur für Visual Basic, sondern für die allermeisten Programmiersprachen gelten. Damit es aber nicht zu theoretisch wird, werden die wichtigsten Passagen durch kleine Visual Basic-Beispiele aufgelockert.*

*In diesem Kapitel lernen Sie auch den Umgang mit Visual Basic-Projekten kennen, der für die Visual Basic-Programmierung sehr wichtig ist.*

Erinnern Sie sich noch an Ihr erstes Schuljahr in der Grundschule? Damals lernten Sie vermutlich beinahe jede Woche einen neuen Buchstaben kennen, aus dem am Anfang kleine Wörter und mit der Zeit immer größere Wörter gebildet wurden. Irgendwann war vielleicht das Y an der Reihe und Sie konnten aus damaliger Sicht so komplizierte Worte wie Xylophon oder Xenophobie bilden (letzteres kam vermutlich erst später an die Reihe). Niemand hätte Ihnen bereits am ersten Tag das komplette Alphabet vorgesetzt und erwartet, dass Sie es am nächsten Tag verstehen. Erst kamen die Buchstaben an die Reihe und Sie lernten mit ihnen umzugehen. Das große Bild ergab sich erst sehr viel später. Beim Erlernen einer Programmiersprache, vor allem wenn es Ihre erste ist, ist es sehr ähnlich. Sie lernen ein oder zwei Befehle kennen und versuchen mit ihnen umzugehen. Dann kommt ein neuer Befehl, mit dem sich wieder neue Möglichkeiten ergeben und Sie komplizierte Wörter buchstabieren, sprich Programmbefehle kombinieren, können. Irgendwann ergibt sich daraus das »große Bild« und Sie haben die Programmierung verstanden. Natürlich gibt es auch Unterschiede. Da wäre zum Beispiel der Umstand, dass Sie bei weitem nicht alle Befehle der Programmiersprache Visual Basic kennen müssen. Von den ca. 90 Befehlen sind nur 10-20 Befehle wichtig, der Rest ist so speziell, dass Sie einige davon auch nach Jahren noch nie benutzt haben werden. Und da wäre der Umstand, dass die »Programmierschule« nicht aus vier, sondern im Prinzip aus unendlich vielen Schuljahren besteht, denn wer sich heutzutage beruflich mit Softwareentwicklung beschäftigt, muss immer wieder etwas Neues lernen. Daran sollten Sie aber im Moment noch nicht denken.

In diesem Kapitel ist das ABC der Programmierung an der Reihe. Sie lernen einfache Befehle der Programmiersprache Visual Basic kennen und machen daraus kleine Programme, die Sie eintippen und über die F5-Taste starten können. Es ist aber in erster Linie ein theoretisches Kapitel, das Sie ruhig mehrfach lesen sollten.

Die Themen für dieses Kapitel:

- Die Rolle der Befehle
- Fehlermeldungen – wenn Visual Basic unzufrieden ist
- Der allgemeine Aufbau eines Programms
- Kommentare
- Variablen
- Computer können gut rechnen
- Entscheidungen

- Programmschleifen
- Einfache Ausgaben über die MsgBox-Funktion
- Einfache Eingaben über die Inputbox-Funktion
- Nützliche Funktionen, die Visual Basic-Programmierer kennen sollten

## Die Rolle der Befehle

Eine Programmiersprache enthält keine Vokabeln, sondern Befehle (engl. »statements« und nicht »commands«, was auch naheliegend gewesen wäre). Ein Befehl ist ein (englisches) Wort (z.B. GoTo oder Input), das für die Programmiersprache eine spezielle Bedeutung hat. Ist die Programmiersprache auf einem PC installiert und tippt der Programmierer das Wort in eine Datei ein, die anschließend abgespeichert und von der Programmiersprache »interpretiert« wird (alle diese Dinge finden bei Visual Basic in der »Entwicklungsumgebung« statt), wird der Befehl ausgeführt. Werden mehrere Befehle aneinandergereiht, ergibt sich ein Programm. Die Befehle eines Programms gehören stets zu einer Programmiersprache, wie z.B. Visual Basic. Es ist im Allgemeinen nicht möglich, die Befehle verschiedener Programmiersprachen in einem Programm zu kombinieren (bei den Windows-Skriptsprachen ist es möglich, doch ist das eine seltene Ausnahme). Ein Programm ist zunächst nur eine Datei (meistens eine normale Textdatei, die eine spezielle Erweiterung besitzt, z.B. .Bas für Basic-Befehlsdatei). Damit das Programm aktiv werden kann, muss es ausgeführt werden (der engl. Ausdruck lautet »to run a programm«). Das bedeutet, der für die Programmiersprache zuständige Interpreter (bei Visual Basic ist er fest eingebaut) arbeitet das Programm Befehl für Befehl ab, sieht sich die einzelnen Befehle an und tut das, was durch den Befehl festgelegt wird.

Damit ein Programm aktiv werden kann, muss es mit einem anderen Programm ausgeführt werden, das die Befehle des Programms interpretiert und in die Befehle des Computers umsetzt. Dieses Programm wird *Interpreter* genannt. Bei Visual Basic ist der Interpreter in die Entwicklungsumgebung eingebaut und wird aktiv, wann immer die F5-Taste gedrückt wird.

Hier ein einfaches Beispiel für einen Programmablauf, der einen Geldautomaten »simuliert«:

```
If PinNumber = 1234 Then
    MsgBox "Die Pin-Nummer ist in Ordnung"
End If
```

Das sind drei Befehlszeilen, die Visual Basic-Befehle enthalten. Die Befehle heißen `If`, `Then` und `End If`. Sie sind dazu da, eine Entscheidung durchzuführen (mehr zu Entscheidungen in Kapitel 5). In der zweiten Befehlszeile steht `Msgbox`. Das ist kein Befehl, sondern eine Funktion. Funktionen sind Befehlen sehr ähnlich, wenngleich es einige Unterschiede gibt, die Sie noch kennen lernen werden. Die Funktion mit dem Namen `Msgbox` ist dazu da, etwas in einer Mitteilungsbox auf dem Bildschirm anzuzeigen. Das, was angezeigt werden soll, folgt in Anführungsstrichen. Die Anführungsstriche sagen Visual Basic »Ignoriere alles, was innerhalb der Anführungsstriche steht«. Das bedeutet, dass Sie in Anführungsstriche alles schreiben können. Da es für Visual Basic keine Bedeutung hat, spielt es auch keine Rolle, wie Sie es schreiben. Wichtig ist nur, dass diese sog. *Zeichenkette* (der englische Name lautet »string«) in Anführungsstriche eingerahmt werden muss. Ansonsten kommt es bereits nach der Eingabe zu einem Syntaxfehler (mehr zu diesem Thema in Kapitel Wenn Visual Basic »meckert«).

Die Aufgabe des `If`-Befehls ist es zu prüfen, ob das, was unmittelbar auf den `If`-Befehl folgt, einen Wert ergibt, der größer als zutreffend ist – Programmierer nennen dies einen Vergleich. Der Vergleich lautet in diesem Fall

```
PinNummer = 1234
```

Doch was ist `PinNummer`? Wie diese Fragestellung bereits dezent andeutet, ist es kein Befehl. Es handelt sich um eine Variable, d.h. um einen (beliebigen) Namen, der für einen (beliebigen) Wert steht. Trifft Visual Basic auf dieses Wort, »denkt« es sich »Hoppla, das ist ja kein Befehl, das muss eine Variable oder Funktion sein«. Es sieht in einem internen »Gedächtnisspeicher« nach und findet dort den Wert dieser Variablen (der zuvor von jemandem eingegeben wurde).

Ist der Wert der Variablen `PinNummer` 1234, dann (und nur dann) ist die Bedingung erfüllt. Man spricht von einer wahren Bedingung. Für die Programmausführung bedeutet dies, dass alle Befehle ausgeführt werden, die auf den `If`-Befehl folgen. Beträgt der Wert der Variablen `PinNummer` dagegen nicht 1234, dann ist die Bedingung nicht erfüllt. Man spricht von einer nicht wahren oder falschen Bedingung (der Begriff »falsch« stellt aber keine Wertung dar, sondern beschreibt lediglich den Zustand). In diesem Fall werden die Befehle zwischen `If` und `End If` nicht ausgeführt, Visual Basic tut als gäbe es sie gar nicht.

Das ist ein Beispiel für die »Denkweise« eines Computers. Sie werden im weiteren Verlauf des Buchs noch manches Mal feststellen, dass Computer sehr viel schlichter strukturiert sind, als Sie es sich vermutlich vorgestellt haben. Computer denken oft sehr einfach. Dadurch, dass sie aber unglaublich

schnell »denken«, erscheint es manchmal so, als wären sie intelligent. Das ist aber bei PCs definitiv nicht der Fall.

## Die Rolle der Syntax

Die schlichte Denkweise der Computer führt dazu, dass sich die Programmierer sehr genau an bestimmte Schreibweisen halten müssen. Diese Schreibweise wird *Syntax* genannt. Wird sie nicht exakt eingehalten, ist ein *Syntaxfehler* die Folge. Ein Syntaxfehler bedeutet, dass der Computer, in diesem Fall Visual Basic, einen Programmbefehl nicht versteht. Nicht, weil dieser zu kompliziert formuliert wurde, sondern weil irgendwo ein Leerzeichen oder ein Komma fehlt oder zuviel ist oder eine andere Formalität übersehen wurde. Das folgende Beispiel sollte Ihnen bekannt vorkommen, denn es ist das Befehlsbeispiel aus dem letzten Absatz:

```
If PinNumber = 1234 Ten
    MsgBox "Die Pin-Nummer ist in Ordnung"
Endif
```

Dieses kleine Beispiel enthält nicht einen, sondern zwei Syntaxfehler. Wenn Sie genau hinsehen, sollten Sie sie erkennen. Fehler Nr. 1 ist schnell gefunden, denn beim Then-Befehl wurde ein H vergessen. Fehler Nr. 2 ist dagegen bereits etwas subtilerer Natur. Zwischen dem End und dem If muss immer ein Leerzeichen stehen.

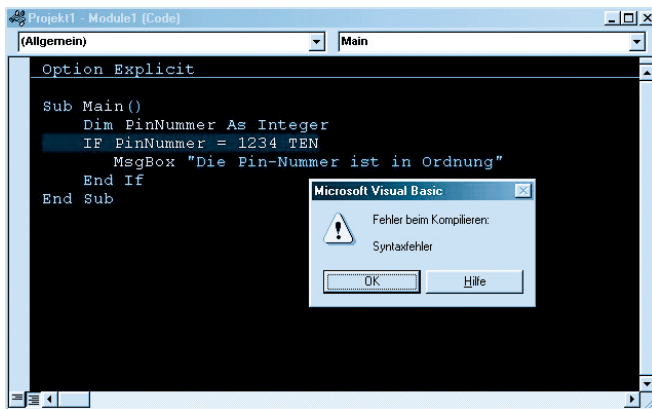


Abbildung 3.1: Der falsche Ten-Befehl führt zu einem Syntaxfehler.

Abbildung 3.1 zeigt, wie Visual Basic einen solchen Syntaxfehler anzeigt. Und warum wird der zweite Fehler nicht ebenfalls angemerkert? Nun, zum einen kann eine Fehlermeldung immer nur einen Fehler beanstanden. Zum anderen hat Visual Basic bereits nach der Eingabe zwischen das End und das

If ein Leerzeichen gesetzt, sodass der vermeintliche Syntaxfehler gar keiner ist. Manchmal denkt Visual Basic bereits bei der Eingabe mit.

## Von der Theorie in die Praxis

Damit Sie die Arbeitsweise eines Computers im Allgemeinen und von Visual Basic im Speziellen besser nachvollziehen können, sollen Sie das kleine Geldautomatenbeispiel aus dem letzten Abschnitt auch einmal in Aktion erleben. Führen Sie zur Umsetzung die folgenden Schritte aus:

**1** Starten Sie *Visual Basic*.

**2** Legen Sie ein STANDARD-EXE-Projekt an. Sollte unmittelbar nach dem Start keine Auswahlbox erscheinen, wurde bereits ein STANDARD-EXE-Projekt angelegt.

**3** Klicken Sie das Formularfenster doppelt an. Sie gelangen damit in das Programmcodefenster und dort in die Ereignisprozedur *Form\_Load*. Das ist die Prozedur, die nach dem Programmstart automatisch aufgerufen wird, bevor das Formular angezeigt wird. Sollen irgendwelche Befehle also gleich mit dem Programmstart ausgeführt werden, müssen sie hier eingefügt werden.

**4** Fügen Sie in die Prozedur *Form\_Load* die folgenden Befehle ein:

```
Dim PinNummer As Integer
PinNummer = Inputbox("Bitte Pin eingeben:")
If PinNummer = 1234 Then
    MsgBox "Die Pin ist richtig!"
Else
    MsgBox "Die Pin ist falsch", 48, "Probieren Sie mal 1234"
End If
```

**5** Starten Sie das Programm über die -Taste.

Unmittelbar nach dem Start sollte eine kleine Eingabebox mit einer Eingabeaufforderung erscheinen. Visual Basic wartet jetzt auf Ihre Eingabe. Geben Sie in das Eingabefeld eine Zahl ein und klicken Sie auf OK. War die Zahl 1234, dann (und nur dann) ist die Bedingung, die durch den If-Befehl geprüft wird, erfüllt. Die Befehle zwischen Then und Else werden ausgeführt. Ansonsten ist die Bedingung nicht erfüllt. Jetzt werden die Befehle zwischen Else und End If ausgeführt. Das ist Computerlogik in Aktion. Mehr zu diesen Dingen in Kapitel 5, in dem es ausschließlich um Entscheidungen geht.

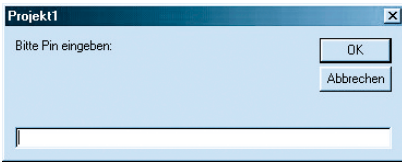





Abbildung 3.2: Nach dem Programmstart erscheint eine Eingabebox.

Wenn Sie die Eingabe wiederholen möchten, müssen Sie das Programm zunächst beenden (z.B. über den **BEENDEN**-Knopf in der Symbolleiste) und anschließend neu starten. Das ist natürlich auf die Dauer sehr umständlich. Sie werden daher spätestens in Kapitel 4 mit der Befehlschaltfläche eine Möglichkeit kennen lernen, wie sich Programmfunktionen beliebig oft auf Knopfdruck wiederholen lassen.

## Wenn Visual Basic »meckert«

Am Anfang wird es Ihnen oft passieren, dass Sie eine Befehlszeile aus dem Buch abtippen oder sich eine Befehlszeile ausdenken, die -Taste drücken und vermutlich nicht gerade angenehm überrascht feststellen, dass Visual Basic mit Ihrer Eingabe nicht zufrieden ist, sondern anfängt zu »meckern« (Programmierer nennen das einen Kompilierfehler). Das liegt daran, dass Sie sich bei der Eingabe einer Befehlszeile an einen bestimmten Aufbau halten müssen und Visual Basic Ihre Eingabe bereits nach dem Drücken der -Taste überprüft (das lässt sich in den Optionen der IDE abstellen). Diese Syntaxfehler haben Sie bereits im letzten Abschnitt kennen gelernt.

Keine Sorge, ein solcher »Fehler« hat keine negativen Konsequenzen und ist schnell korrigiert. Sie müssen lediglich den **OK**-Button klicken (das Anklicken des **HILFE**-Buttons bringt im Allgemeinen nicht viel), den Fehler korrigieren und es erneut probieren. War alles richtig, beruhigt sich Visual Basic wieder und alles ist in Ordnung.

Es gibt aber auch Fehler, die treten erst während des Programmverlaufs auf, also nachdem das Programm über die -Taste gestartet wurde. Zuerst geht alles gut, doch dann passiert es. Visual Basic hält die Programmausführung an und beschwert sich über eine bestimmte Befehlszeile. Ein solcher Fehler wird *Laufzeitfehler* (engl. »run time error«) genannt, da er erst während der Laufzeit des Programms (also während das Programm läuft) auftritt. Auch Laufzeitfehler sind kein Grund zur Beunruhigung, sie besitzen nur andere Ursachen.

Tritt ein Laufzeitfehler auf, bietet Ihnen Visual Basic drei Möglichkeiten an:

1. Das Programm zu beenden.
2. Das Programm zu »debuggen«. Das bedeutet lediglich, dass das Programm in den Haltemodus geschaltet wird.
3. Eine Hilfe abzurufen.

Das Programm zu beenden ist stets die einfachste Variante. Sie können sich den Programmtext in Ruhe ansehen, den Fehler gegebenenfalls korrigieren und das Programm anschließend erneut starten. Hinweise dazu, wo der oder die Fehler zu finden sein könnten, gibt Ihnen Visual Basic allerdings nicht. Es ist in erster Linie die Erfahrung, das gewisse Fingerspitzengefühl und manchmal auch wenig Glück, die für die erfolgreiche Fehlersuche in großen Programmen ausschlaggebend sind.

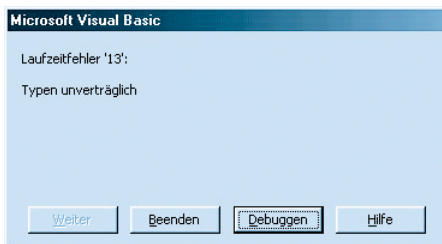


Abbildung 3.3: BEENDEN, DEBUGGEN, HILFE oder was? Tritt ein Laufzeitfehler auf, gibt es im Allgemeinen drei Möglichkeiten, wie es weiter geht.

Die zweite Variante ist das Debuggen eines Programms. Dieser Begriff wird in Kapitel 10 noch etwas ausführlicher vorgestellt. Im Moment soll der Hinweis genügen, dass das Programm durch Anklicken der DEBUGGEN-Schaltfläche in den Haltemodus übergeht. Und es kommt noch besser. Visual Basic zeigt Ihnen jene Programmzeile an, die den Laufzeitfehler verursacht hat (in der Regel ist sie gelb unterlegt). Sie können sich jetzt die Befehlszeile in Ruhe anschauen, sich im Direktfenster die Werte von Variablen oder beteiligten Eigenschaften ausgeben lassen und einiges mehr. Sie können bis zu einem gewissen Umfang sogar Änderungen am Programmtext vornehmen. Sollte durch eine Änderung ein Neustart des Projekts (nicht von Visual Basic) notwendig werden, erhalten Sie eine entsprechende Meldung. Wenn Sie das Dialogfeld mit OK bestätigen, wird die Änderung zwar durchgeführt, aber das Projekt geht in den Entwurfmodus über. Lehnen Sie die Änderung durch Anklicken von ABBRECHEN ab, wird die Änderung wieder rückgängig gemacht.



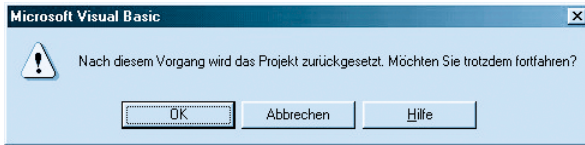


Abbildung 3.4: Diese Meldung erscheint, wenn eine gemachte Änderung dazu führt, dass das Projekt zurückgesetzt wird.

Als dritte Variante bietet Ihnen Visual Basic den Hilfefknopf an. Versprechen Sie sich aber nicht zu viel davon, denn die Hilfetexte sind sehr allgemein gehalten.

### Tip

Starten Sie das Programm nicht mit **F5**, sondern mit **Strg + F5**. Der Unterschied besteht darin, dass in diesem Fall das gesamte Programm kompiliert wird und nicht nur die ersten Zeilen. Diese bedingte Kompilierung kann durch Abwahl der Option **BEI BEDARF** in der Registerkarte **ALLGEMEIN** der **OPTIONEN** deaktiviert werden. Dann haben die **F5**-Taste und **Strg + F5** die gleiche Wirkung.

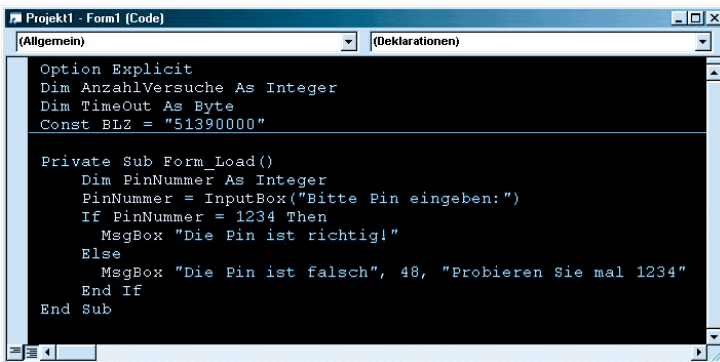
## Der allgemeine Aufbau eines Visual Basic-Programms

Sie wissen bereits, ein Programm besteht aus Befehlen. Doch ganz so einfach ist es bei Visual Basic nicht. Bei Visual Basic arbeiten Sie nicht mit Programmen, sondern mit Projekten. Aus diesem Grund hätte die Überschrift besser »Der allgemeine Projektaufbau« heißen sollen, doch hätte sich dies unnötig kompliziert angehört. Der Begriff *Projekt* ist bei Visual Basic mit dem Begriff *Programm* gleichbedeutend. Jedes Projekt besteht aus Formularen, Modulen, Klassenmodulen usw. Wenn Sie nach dem Start von Visual Basic ein **STANDARD-EXE-Projekt** wählen, erhalten Sie ein noch leeres Formular. Auch wenn ein Visual Basic-Projekt kein Formular enthalten muss, ist dies bei allen Programmen, die Sie in diesem Buch kennen lernen werden, der Fall. Es kommt daher in erster Linie darauf an, den Aufbau eines Formulars zu kennen. Dieser ist sehr einfach, denn ein Formular besteht aus zwei Bereichen: Die Formularoberfläche, auf der die Steuerelemente angeordnet werden, und das Programmcodefenster. Hier findet die Programmierung statt, d.h. hier werden die Befehle eingegeben, die nach dem Programmstart

ausgeführt werden sollen. Der Programmbereich eines Formulars besteht wieder aus zwei Bereichen:

- dem *Allgemein*-Teil und
- den Ereignisprozeduren.

Der *Allgemein*-Teil enthält jene »Vereinbarungen«, die für das gesamte Formular gelten sollen. Wird hier z.B. eine Variable mit dem `Dim`-Befehl deklariert, ist diese Variable in allen Prozeduren des Formulars bekannt. Der *Allgemein*-Teil kann auch Konstantendefinitionen (über den `Const`-Befehl), den Befehl `Option Explicit` sowie Prozedur- und Funktionsdefinitionen enthalten. Einzelne Befehle sind, mit Ausnahme der aufgezählten Befehle, im *Allgemein*-Teil allerdings nicht erlaubt. Der *Allgemein*-Teil eines Formulars wird angezeigt, indem Sie im Programmcodefenster in der linken Auswahlliste den Eintrag (ALLGEMEIN) wählen. Die direkte Auswahl ist allerdings nicht unbedingt notwendig, denn Sie können von der Stelle des Programmcodefensters nach oben scrollen und gelangen so automatisch in den *Allgemein*-Teil. Es gibt im Programmcodefenster keine festen Unterteilungen. Die Trennlinie, die oft unmittelbar nach dem `Option Explicit`-Befehl angezeigt wird, soll lediglich der Orientierung dienen. Sie hat keine echte Funktion und kann auch nicht verschoben werden (lassen Sie sich also durch diese Linie nicht irritieren).



```
Option Explicit
Dim AnzahlVersuche As Integer
Dim TimeOut As Byte
Const BLZ = "51390000"

Private Sub Form_Load()
    Dim PinNummer As Integer
    PinNummer = InputBox("Bitte Pin eingeben:")
    If PinNummer = 1234 Then
        MsgBox "Die Pin ist richtig!"
    Else
        MsgBox "Die Pin ist falsch", 48, "Probieren Sie mal 1234"
    End If
End Sub
```

Abbildung 3.5: Im Allgemein-Teil eines Formulars werden Konstanten und Variablen deklariert.

Sie können ein Visual Basic-Projekt jederzeit um neue Formulare, Module usw. erweitern, indem Sie im PROJEKT-Menü das entsprechende Kommando auswählen. Jedes neue Modul wird im Projekt-Explorer mit seinem Namen angezeigt, den es von Visual Basic erhalten hat, und der über das Eigen-

schaftenfenster ( `F4` -Taste) geändert werden kann (und auch geändert werden sollte).

### Alles spielt sich in Prozeduren ab

Visual Basic besitzt als Programmiersprache eine wichtige Eigenheit. Alle Befehle, mit Ausnahme jener Befehle, die im *Allgemein*-Teil zugelassen sind, befinden sich immer innerhalb einer Prozedur (oder Funktion). Eine Prozedur ist ein Programmbereich, der durch einen Sub-Befehl gefolgt von dem Namen der Prozedur eingeleitet und durch einen `End Sub`-Befehl wieder beendet wird. Hier ein Beispiel für eine Prozedur:

```
Sub AllesNeu ()
```

```
End Sub
```

Der Sub-Befehl definiert eine Prozedur mit dem Namen `AlllesNeu`. Der `End Sub`-Befehl beendet die Prozedur. Diese beiden Befehle definieren den *Rahmen* der Prozedur. Welche Befehle innerhalb einer Prozedur stehen, spielt keine Rolle.

### Spezialfall Ereignisprozedur

In den ersten Beispielen in diesem Buch werden Sie lediglich Ereignisprozeduren kennen lernen. Das sind Prozeduren, deren Rahmen Sie nicht eingeben müssen (und auch nicht sollen), und die von Visual Basic immer dann aufgerufen werden, wenn das betreffende Ereignis, etwa ein Anklicken einer Schaltfläche mit der Maus, eintritt. Ereignisprozeduren unterscheiden sich von normalen Prozeduren nicht nur in ihrem Aufbau, sondern dadurch, dass sie von Visual Basic automatisch aufgerufen werden.

### Privat(e) oder öffentlich?

Das ist bei Visual Basic keine Frage des Geschmacks, sondern bestimmt, ob eine Prozedur nur innerhalb des Formulars (oder allgemein Moduls) oder auch von anderen Modulen des Projekts aufgerufen werden darf. Das ist bereits ein fortgeschritteneres Thema der Visual Basic-Programmierung und wird an dieser Stelle auch nur deswegen erwähnt, weil Ereignisprozeduren das Wort `Private` vorausgeht:

```
Private Sub Form_Load ()
```

```
End Sub
```

Es handelt sich hier um eine Prozedur, die nur innerhalb des Formulars aufgerufen werden kann. Man spricht hier von einer privaten Prozedur. Das Gegenstück ist eine öffentliche Prozedur. Hier geht dem Sub-Befehl entweder das Wort `Public` voraus oder es wird gar nicht aufgeführt, da Visual Basic in diesem Fall ebenfalls von einer öffentlichen Prozedur ausgeht. Die Prozedur

```
Sub AllesNeu ()
```

```
End Sub
```

ist daher öffentlich, was bedeutet, dass sie für andere Module des Projekts »sichtbar« ist. Soll eine Prozedur z.B. von mehreren Formularen aus aufgerufen werden, muss sie Teil eines Moduls sein, das über das Menükommando PROJEKT/MODUL HINZUFÜGEN zum Projekt hinzugefügt wurde.

## Kommentare

Niemand erwartet, dass Sie zu allem und jedem einen passenden Kommentar abgeben. Kommentare besitzen bei einer Programmiersprache dennoch eine wichtige Funktion. Sie dienen dazu, den Quelltext eines Programms zu beschreiben. Bei kleinen Programmen kommt dies seltener vor. Doch stellen Sie sich vor, Sie arbeiten als Programmierer(in) in einem großen Team, in dem auch andere Programmierer Ihren »Quellcode« (so werden die Programmbeefehle auch bezeichnet) lesen und vor allem verstehen müssen. Oder Sie übergeben Ihren Quellcode an eine Urlaubsvertretung oder einen Nachfolger. Dann sind Kommentare ebenfalls unverlässlich. Ein Kommentar ist eine Anmerkung im Quelltext, die eine bestimmte Programmstelle beschreibt. Jede Kommentarzeile wird durch ein Apostroph eingeleitet. Alles, was in der Zeile folgt ist für Visual Basic tabu, d.h. es versucht nicht, daraus einen Reim zu machen (Sie erhalten auch keine Fehlermeldung):

```
' Ich weiss gar nicht mehr so genau was das alles soll
```

## Das Gedächtnis des Computers – Variablen

Für die meisten Programme ist es sehr wichtig, dass sich das Programm Zahlen, Namen und andere Dinge merken kann. Stellen Sie sich ein Buchhaltungsprogramm vor, das nach der Eingabe einer Buchung alle eingegebenen Zahlen gleich wieder vergisst. Ein solches Programm wäre nicht mehr als eine Kuriosität, für die Praxis leider nicht zu gebrauchen. Alle Programmiersprachen ermöglichen daher das Ablegen von Werten in Variablen. Eine

*Variable* ist ein Name, der während der Programmausführung für einen Wert steht, der zuvor in der Variablen gespeichert wurde.

Hinter einer Variablen steht eine Speicherzelle im riesigen Datenspeicher des PCs (dem RAM). Wo genau sich diese Speicherzelle befindet, spielt für die Visual Basic-Programmierung keine Rolle. Es ist eine Art Geheimnis, das sich Visual Basic und Windows teilen (und das hartnäckige Visual Basic-Programmierer über die Funktion `VarPtr` herausbekommen können).

### Variablen müssen deklariert werden

Programmierer lieben es bekanntlich, sich kompliziert auszudrücken. Aus diesem Grund heißt es (vermutlich) auch nicht, dass Variablen bekannt gemacht werden, sie müssen vielmehr deklariert werden. Bei der Deklaration einer Variablen teilt man Visual Basic mit, dass ein Name ab jetzt die Rolle einer Variablen spielen soll. Trifft Visual Basic im weiteren Programmverlauf auf diesen Namen, weiß es, dass es sich hier um eine Variable handelt. Für die Deklaration einer Variablen gibt es bei Visual Basic nicht einen, sondern mit `Dim`, `ReDim`, `Public`, `Private` und `Static` gleich fünf Befehle, die alle ihre Berechtigung haben. In diesem Buch lernen Sie aber nur den `Dim`-Befehl kennen, der für die meisten Zwecke geeignet ist.

Der folgende Befehl deklariert eine Variable mit dem Namen Guthaben:

```
Dim Guthaben As Integer
```

Auf den `Dim`-Befehl folgt immer der Name einer Variablen, den Sie sich frei ausdenken können. Allerdings gelten bestimmte Regeln. So dürfen Variablenamen keine Leerzeichen enthalten und eine bestimmte Länge (die allerdings offenbar nirgendwo in der Visual Basic-Hilfe verraten wird) nicht überschreiten (das sollte aber kein Problem sein). Auch dürfen bestimmte Sonderzeichen (wie z.B. `&`, `$`, `!`, `#` und `@`) nicht vorkommen, da diese eine spezielle Bedeutung haben. Umlaute sind dagegen erlaubt. Die Groß- und Kleinschreibung spielt ebenfalls keine Rolle. Sie werden vielmehr feststellen, dass Visual Basic nach der Eingabe die Schreibweise so anpasst, wie sie bei der Deklaration festgelegt wurde.

### Erst deklarieren, dann zuweisen

Mit der Deklaration wird eine Variable lediglich bekannt gemacht. Ihren Wert erhält sie erst später:

```
Dim Guthaben As Single
```

```
Guthaben = -6567,89
```

Das Zuweisen eines Wertes an eine Variable wird *Zuweisung* genannt (liegt irgendwie nahe). Es ist bei Visual Basic 6.0 nicht möglich, die Zuweisung mit der Deklaration zu kombinieren.

## Die Rolle des Datentyps

Damit es nicht zu einfach wird (das ist natürlich ironisch gemeint), muss (besser sollte) bei der Deklaration einer Variablen auch der *Datentyp* angegeben werden. Er legt fest, welche Sorte von Daten in der Variablen später abgelegt werden dürfen. Es sind also nicht alle Variablen gleich. Das ist in etwa so, als würden Sie sich vor dem Backen eine Kuchenform aussuchen. Sie legt fest, welche Form der Kuchen später hat. Statt Herz, Ente oder Stern sind die »Kuchenformen« bei den Variablen *Integer*, *Long*, *Single*, *Currency* oder *String*. Soll eine Variable nur ganze Zahlen aufnehmen, nimmt man *Integer*, soll sie auch Zahlen mit einem Nachkommanteil aufnehmen *Single* und wenn es Zeichenketten sein sollen, dann *String*. Insgesamt gibt es 12 verschiedene Datentypen, die Sie am Anfang aber nicht alle kennen müssen. Damit Ihnen die Auswahl nicht so schwer fällt, präsentiert Ihnen Visual Basic nach Eingabe von *As* freundlicherweise eine Liste mit allen in Frage kommenden Typen. Sind das nicht mehr als 12? Ja, weil in der Auswahlliste auch alle Klassennamen enthalten sind, die ebenfalls für die Deklaration einer Variablen in Frage kommen. Die wichtigsten Datentypen sind in Tabelle 3.1 zusammengestellt, eine komplette Aufstellung finden Sie im Anhang.

Datentyp	Wozu ist er gut?
Currency	Für Zahlen, bei denen die Genauigkeit auf vier Stellen eine wichtige Rolle spielt, z.B. bei Geldbeträgen.
Integer	Für ganze Zahlen, die im Bereich -32.768 bis +32.767 liegen können.
Long	Für ganze Zahlen, die im Bereich -2.147.483.648 bis + 2.147.483.647 liegen können.
String	Für Zeichenketten (also Texte).

Tabelle 3.1: Die wichtigsten Datentypen bei Visual Basic, die Sie am Anfang kennen sollten.

Zwei Anmerkungen zum Schluss: Wird bei einer Deklaration einer Variablen kein Datentyp angegeben, verwendet Visual Basic immer den Datentyp *Variant*. Dies ist der »Oberdatentyp«, der alle anderen Datentypen um-

fasst. Dennoch sollte man sich die Mühe machen und Datentypen explizit angeben. Ein `Dim`-Befehl kann auch mehrere Variablen deklarieren:

```
Dim iAnzahlPunkte As Integer, nAnzahlTreffer As Long
```

Der Datentyp muss allerdings für jede Variable einzeln angegeben werden. Eine beliebige Quiz-Frage im Einstellungstest bei Visual Basic-Programmierern in großen Softwarefirmen lautet: Welchen Datentyp besitzt die Variable `A` nach der folgenden Deklaration:

```
Dim A, B, C As Integer
```

Die Antwort lautet `Variant`, da weder für `A` noch für `B` ein Datentyp angegeben wurde. Auch wenn dies im Kapitel 2 eines Einführungsbuchs noch zu weit weg ist, sollte Sie zumindestens diese Frage nicht mehr in die vorgesehene Falle locken (bei der nächsten Visual Basic-Version `VB.Net` wird dies allerdings nicht mehr gelten).

## Konstanten – wenn alles gleich bleiben soll

Neben Variablen gibt es – wie in den meisten Programmiersprachen auch – bei Visual Basic die *Konstanten*. Es handelt sich um Variablen, deren Wert sich nicht ändern kann. Konstanten bieten gegenüber der direkten Eingabe des Wertes, für den sie stehen sollen, zwei wichtige Vorteile:

- Ändert sich der Wert, muss nur die Konstante geändert werden. Überall dort, wo im Programm der Konstantenname auftaucht, wird der neue Wert eingesetzt.
- Der Konstantenname ist oft kürzer als der eigentliche Wert, was vor allem bei Zeichenketten praktisch ist (denken Sie an lange Verzeichnisnamen).

Konstanten sind für die Programmierer deswegen wichtig, weil sie sie benutzen können ohne sich darüber Gedanken machen zu müssen, welchen Wert sie besitzen, oder ob ein anderer Programmteil den Wert verändert haben könnte. In kleineren Visual Basic-Programmen werden Konstanten nur selten benötigt.

Eine Konstante wird mit dem `Const`-Befehl deklariert:

```
Const MwstSatz1 = 16
```

Anders als bei einer Variablen erhält die Konstante bereits mit der Deklaration ihren Wert, denn eine spätere Zuweisung ist nicht möglich.

Konstanten sind vor allem praktisch, wenn es um Verzeichnisnamen geht:

```
Const Pfad = "C:\Eigene Dateien\Eigene Programme\VB\"
```

Es ist sehr viel kürzer in einem Befehl die Konstante Pfad anzugeben als den Wert, für den die Konstante steht.

### Hinweis

*Es ist üblich, Konstanten in Großbuchstaben zu schreiben. Diese Konvention spielt bei Visual Basic allerdings keine wichtige Rolle.*

## Visual Basic kennt viele Konstanten

Bei Visual Basic sind viele Konstanten bereits »fest eingebaut«. Wundern Sie sich daher nicht, wenn in einem Beispiel Namen auftauchen, die scheinbar nirgendwo deklariert wurden. Beispiele dafür sind `vbCrLf`, das für die ASCII-Codes 13 und 10 und damit für einen Zeilenumbruch steht, oder die Farbkonstanten, wie `vbRed` oder `vbBlack`. Insgesamt kennt Visual Basic weit über 100 Konstanten.

### Tipp

*Möchten Sie die Konstanten sehen, die Visual Basic kennt, dann drücken Sie die `[F2]`-Taste (das öffnet den Objektkatalog in Visual Basic) und wählen aus der oberen Auswahlliste den Eintrag `VBRUN`. Wenn Sie in der Liste der Klassen scrollen, sehen Sie die verschiedenen Gruppen von Konstanten (z.B.: `COLORCONSTANTS`), die an dem für sie typischen Symbol (den beiden nebeneinander liegenden gelben Chips) zu erkennen sind.*

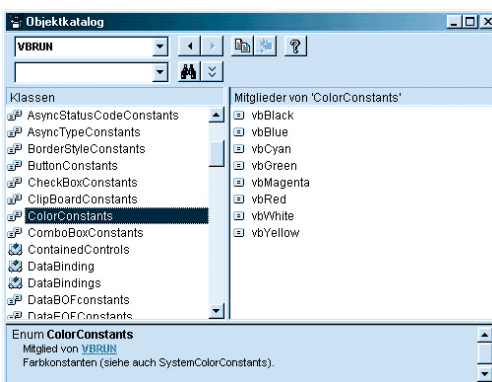


Abbildung 3.6: Der Objektkatalog zeigt die in Visual Basic fest eingebauten Konstanten an.



## Wieviele Tage sind Sie denn alt?

Das folgende Beispielprogramm soll die Rolle von Variablen in einem Visual Basic-Programm veranschaulichen. Nach dem Start geben Sie in die Eingabebox Ihr Geburtsdatum ein. Nach Anklicken der Befehlsschaltfläche wird Ihr Alter in Tagen angezeigt. Im Mittelpunkt des kleinen Programms steht eine Variable, die sich das eingegebene Datum »merkt«, damit das Programm im nächsten Schritt die Anzahl an Tagen ausrechnen kann.

Führen Sie zur Umsetzung die folgenden Schritte aus:

- 1** Starten Sie *Visual Basic*.
- 2** Legen Sie ein STANDARD-EXE-Projekt an. Sollte unmittelbar nach dem Start keine Auswahlbox erscheinen, wurde bereits ein STANDARD-EXE-Projekt angelegt.
- 3** Klicken Sie das Formularfenster doppelt an. Sie gelangen damit in das Programmcodefenster und dort in die Ereignisprozedur *Form\_Load*. Das ist die Prozedur, die nach dem Programmstart automatisch aufgerufen wird.
- 4** Fügen Sie in die Prozedur *Form\_Load* die folgenden Befehle ein:

```
Dim Geburtstag As Date
Dim AnzahlTage As Integer
Geburtstag = InputBox("Bitte das Geburtsdatum eingeben:")
AnzahlTage = DateDiff("d", Geburtstag, Now)
MsgBox "Sie sind " & AnzahlTage & " alt"
```

- 5** Starten Sie das Programm über die **F5**-Taste (falls eine Abfrage für das Programm erscheint, klicken Sie auf die NEIN-Schaltfläche).

Nach dem Start erscheint eine Eingabebox, in die Sie ein Datum eingeben sollen. Nach Anklicken von OK wird diese Eingabe in der Variablen mit dem Namen *Geburtstag* (der Name spielt dabei keine Rolle) gespeichert. Im nächsten Schritt wird der Wert dieser Variablen von der *DateDiff*-Funktion in Visual Basic verrechnet und das Ergebnis wird in einer weiteren Variablen, sie heißt *AnzahlTage*, gespeichert. Der Wert dieser Variablen wird von der folgenden *Msgbox*-Funktion in einer kleinen Mitteilungsbox angezeigt. Die Variablen dienen dazu, die Werte, mit denen das Programm arbeiten soll, aufzubewahren. Wird das Programm beendet, sind auch die Variablen mit ihren Werten weg.

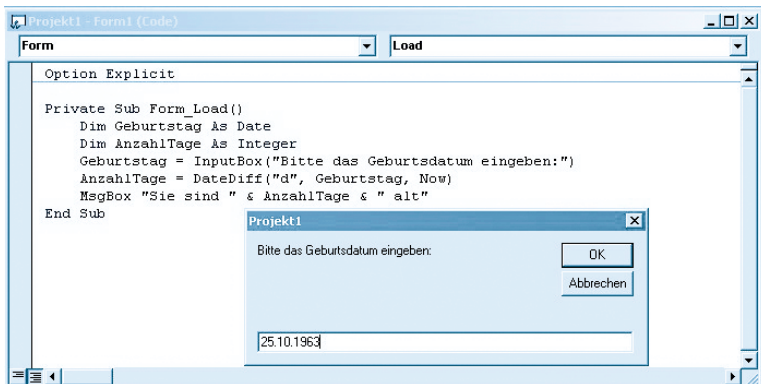


Abbildung 3.7: Der über die Eingabebox eingegebene Wert wird in einer Variablen gespeichert, umgerechnet und wieder ausgegeben.

Sollte das Programm eine Fehlermeldung anzeigen (z.B. »Typen unverträglich«), liegt dies höchstwahrscheinlich daran, dass das Datum nicht das richtige Format (richtig wäre z.B. »25.10.1963«) besitzt. Klicken Sie auf BEENDEN und starten Sie das Programm noch einmal.

## Ein Computer kann auch rechnen

Sie werden es kaum glauben, aber ein Computer kann natürlich auch rechnen. Das ist schon immer seine Hauptdaseinsberechtigung und diesem Umstand verdankt er auch seinen Namen (to compute, engl. »rechnen«). Es gibt allerdings keinen Rechenbefehl in Visual Basic. Berechnungen werden vielmehr mit Hilfe von Operatoren und Funktionen durchgeführt. Eine Aufstellung aller Operatoren und Funktionen finden Sie im Anhang. Die folgenden kleinen Beispiele sollen nur das Prinzip von Berechnungen deutlich machen.

Eine Berechnung setzt sich stets aus Zahlen, Variablen und Operatoren zusammen, wobei wie bei einer richtigen Formel bestimmte Regeln eingehalten werden müssen. In den meisten Fällen wird das Ergebnis einer Berechnung einer Variablen zugewiesen. Der folgende Befehl führt eine einfache Addition durch:

```
iSumme = 4 + 5
```

Das Ergebnis dieser Berechnung wird nicht angezeigt, sondern der Variablen `iSumme` zugewiesen, die jetzt den Wert 9 besitzt. Natürlich ist das eine sehr einfache Berechnung, Visual Basic kann selbstverständlich etwas mehr:

```
sMwst = BruttoBetrag / 100 * 16
```

oder

```
sFormelX = Sqr(Kraft * Masse) ^2 / (Lichtgeschw / (1 -  
Solarkonstante))
```

Bei dieser Berechnung sind bereits deutlich mehr Operatoren involviert. Außerdem mussten Klammern gesetzt werden, weil auch bei Visual Basic die bekannte Punkt-Vor-Strich-Rechenregel gilt.

Nicht alle Berechnungen lassen sich in einer einzigen Zeile unterbringen. Berechnungen, bei denen eine zeitliche Varianz im Spiel ist (die entsprechend der Länge eines Zeitraums mehrfach wiederholt werden), können nicht mit einem Befehl durchgeführt werden. Hier kommen Programmschleifen ins Spiel. Ein erstes Beispiel lernen Sie in Kapitel 6 kennen, in dem ein Geldbetrag über eine vorgegebene Anzahl an Jahren erhöht wird, um den Zinsertrag für jedes Jahr ausgeben zu können.

Hier eine Kostprobe für die Rechenkünste eines Visual Basic-Programms: Das folgende kleine Programm berechnet den Kreisumfang nach der bekannten Formel  $u = \pi * r^2$ . Dazu muss man allerdings wissen, dass Visual Basic die Kreiszahl  $\pi$  gar nicht kennt und diese durch eine Konstante hinzugefügt werden muss:

```
Const Pi = 3.14  
Dim snRadius As Single  
Dim snUmfang As Single  
snRadius = 2.5  
snUmfang = Pi * snRadius ^2  
Msgbox "Der Kreisumfang beträgt: " & snUmfang
```

Dieses Beispiel macht deutlich, dass selbst ein Computer simple Rechenoperationen nicht in einem Schritt erledigen kann, sondern dass dazu stets mehrere Schritte aneinandergereiht werden müssen.

## Der Computer trifft Entscheidungen

Entscheidungen gehören zu den wichtigsten Befehlen einer Programmiersprache. Da Sie mit dem `If`-Befehl zu Beginn dieses Kapitels (als es um die Frage ging, wie »denkt« ein Computer) bereits das Prinzip der Entscheidungen kennen gelernt haben und alle Details in Kapitel 5 an der Reihe sind, gibt es in diesem Abschnitt nur eine kurze Zusammenfassung.

Wann immer ein Programm eine Entscheidung treffen muss, basiert diese auf einem simplen Vergleich. Dieser lautet: Ist der Wert eines Ausdrucks größer Null (also wahr) oder ist er Null (und damit falsch). Auf diesem einfachen Prinzip basieren alle Entscheidungen, die bei der Ausführung eines Computerprogramms getroffen werden.

## Ein Beispiel für eine sehr einfache Entscheidung

Da es erfahrungsgemäß eine Weile dauert bis angehende Programmierer mit den Grundgesetzen der binären (weil auf den beiden Zuständen wahr und falsch basierenden) Logik umgehen können, sollen im Folgenden anstelle weiterer Erläuterungen ein paar Beispiele auf binäre Denkmuster vorbereiten. Das erste Beispiel führt eine simple Überprüfung durch. Sie geben nach dem Programmstart Ihr Geburtsdatum ein und der Computer teilt Ihnen mit, ob Sie bereits volljährig sind oder nicht.

Führen Sie zur Umsetzung die folgenden Schritte aus:

**1** Starten Sie *Visual Basic*.

**2** Legen Sie ein STANDARD-EXE-Projekt an. Sollte unmittelbar nach dem Start keine Auswahlbox erscheinen, wurde bereits ein STANDARD-EXE-Projekt angelegt.

**3** Klicken Sie das Formularfenster doppelt an. Sie gelangen damit in das Programmcodefenster und dort in die Ereignisprozedur *Form\_Load*. Das ist die Prozedur, die nach dem Programmstart automatisch aufgerufen wird.

**4** Fügen Sie in die Prozedur *Form\_Load* die folgenden Befehle ein:

```
Dim Alter As Integer
Alter = InputBox("Bitte Ihr Alter eingeben:")
If Alter > 18 Then
    MsgBox "Sie sind volljährig!"
Else
    MsgBox "Sie sind leider noch nicht volljährig!"
End If
```

**5** Starten Sie das Programm über die **F5**-Taste (falls eine Abfrage für das Programm erscheint, klicken Sie auf die NEIN-Schaltfläche).

Nach dem Start des Programms erscheint eine kleine Eingabebox, in der Sie Ihr Alter eingeben müssen. Nach Bestätigen mit OK erhalten Sie vom Computer die Auskunft darüber, ob Sie nun volljährig sind oder nicht. Natürlich benötigt man dazu keine Computer, das Beispiel soll lediglich als Anschauungsbeispiel für die »Denkweise« eines Computerprogramms dienen. Das Programm kennt weder Ihr wahres Alter noch hat der Begriff Volljährigkeit irgendeine Bedeutung. Es führt lediglich einen simplen Vergleich zweier Zahlen durch, der aber im Kontext, in dem das Programm eingesetzt wird, als Entscheidung eines Computerprogramms interpretiert werden könnte.

Das letzte Beispiel war wirklich sehr simpel. Etwas realistischer wäre das Programm, wenn es ein Geburtsdatum entgegennehmen und anhand des

aktuellen Datums ermitteln würde, ob die Volljährigkeit schon erreicht ist. Nun, das ist natürlich kein Problem. Ein Computerprogramm ist immer nur so gut wie es die Befehle, aus denen es besteht, zulassen. Wenn Sie die Befehle in *Form\_Load* durch die folgenden Befehle austauschen, werden Sie nach dem Start nach Ihrem Geburtsdatum gefragt:

```
Dim Geburtstag As Date
  Geburtstag = InputBox("Bitte das Geburtsdatum eingeben:")
If DateDiff("yyyy", Geburtstag, Date) > 18 Then
  MsgBox "Sie sind volljährig!"
Else
  MsgBox "Sie sind leider noch nicht volljährig!"
End If
```

Das obige Programm ist natürlich alles andere als perfekt. Würden Sie beispielsweise am 31.12.2002 volljährig werden und das Programm am 1.1.2002 ausführen, wären Sie für das Programm bereits volljährig, da es über die *DateDiff*-Funktion lediglich die Differenz der Jahreszahlen ausrechnet. Soll das Programm wirklich richtig rechnen, müssen die Entscheidungsregeln etwas verfeinert werden. Neben der Jahreszahl müssen auch der Monat und der Tag verglichen werden. Der neue Inhalt der *Form\_Load*-Prozedur muss wie folgt aussehen:

```
Dim Geburtstag As Date
Geburtstag = InputBox("Bitte das Geburtsdatum eingeben:")
If Year(Date) - Year(Geburtstag) >= 18 _
  And Month(Date) >= Month(Geburtstag) _
  And Day(Date) >= Day(Geburtstag) Then
  MsgBox "Sie sind volljährig!"
Else
  MsgBox "Sie sind leider noch nicht volljährig!"
End If
End
```

Jetzt erfährt man auf den Tag genau, ob man endlich volljährig ist oder nicht (sollten Sie dieses wichtige Ereignis bereits hinter sich gelassen haben, denken Sie sich ein anderes Datum aus, um das Programm zu testen). Über den *And*-Operator werden insgesamt drei Vergleiche kombiniert. Auch dieser »Monsterausdruck«, der dank des Zeilenfortführungsoperators auf mehrere Zeilen verteilt wurde, ergibt lediglich einen einzigen Wert, der entweder wahr oder falsch ist. Aus diesem kleinen Beispiel lassen sich zwei wichtige Lehren ziehen: Ein Computerprogramm kann immer nur so gut sein, wie es durch die Programmierer vorgegeben wurde. Und, Programmierer haben es nicht leicht, denn scheinbar simple Aufgaben können manchmal ganz schön knifflig zu lösen sein.

## Wenn Computer »schummeln«

Das letzte Beispiel in diesem Abschnitt soll den Beweis antreten, dass Computer keine Skrupel haben zu schummeln. Natürlich nur dann, wenn es durch das Programm so vorgesehen ist.

Das folgende Programm spielt Geldautomat und erlaubt ausgehend von einem (natürlich fiktiven) Guthaben von 1.500 DM die Auszahlung von Geldbeträgen (max. sind 400 DM erlaubt). Heimlich (was Geldautomaten natürlich nie machen) zweigt es sich bei jeder Buchung 50 DM ab, zeigt aber die scheinbar echte, in Wirklichkeit aber falsche Höhe des Guthabens an. Die Wahrheit kommt ans Licht, wenn versucht wird, auf der Grundlage des fiktiven Guthabens einen Betrag abzuheben, der wahre Betrag aber für eine Abhebung nicht mehr ausreicht.

Das folgende Beispiel ist bereits etwas umfangreicher und nimmt einige Dinge vorweg, die erst in den folgenden Kapiteln erklärt werden. Probieren Sie es dennoch einmal aus, da die Umsetzung trotzdem nicht sehr schwierig ist.

- 1 Starten Sie *Visual Basic*.
- 2 Legen Sie ein STANDARD-EXE-Projekt an. Sollte unmittelbar nach dem Start keine Auswahlbox erscheinen, wurde bereits ein STANDARD-EXE-Projekt angelegt.
- 3 Klicken Sie das Formularfenster doppelt an. Sie gelangen damit in das Programmcodefenster und dort in die Ereignisprozedur *Form\_Load*. Das ist die Prozedur, die nach dem Programmstart automatisch aufgerufen wird.
- 4 Geben Sie in die Prozedur den folgenden Befehl ein:  
Guthaben = 1500
- 5 Durch diesen Befehl wird das Startguthaben festgelegt.
- 6 Wählen Sie aus der linken Auswahlliste den Eintrag (ALLGEMEIN). Sie wählen damit den *Allgemein*-Teil des Formulars aus. Geben Sie dort folgende Befehle ein:  
Private Guthaben As Integer  
Private Schummelkonto As Integer  
Private Auszahlungsbetrag As Integer
- 7 Ordnen Sie auf dem Formular ein Textfeld an. Sie finden es in der Werkzeugsammlung ganz oben (es enthält als Inschrift »ab«). Die Werkzeugsammlung ist das kleine, schmale Fenster mit den vielen Bildchen. Klicken Sie das Symbol einmal an, lassen Sie den Mauszeiger wieder los, bewegen Sie den Mauszeiger auf eine freie Stelle auf dem Formular und spannen Sie bei gedrückter linker Maustaste einen Rahmen auf. Sie haben damit auf dem Formular ein Textfeld angeordnet.

**8** Ordnen Sie auf dem Formular eine Schaltfläche an. Sie finden sie in der Werkzeugsammlung unterhalb des Textfelds.

Klicken Sie die Schaltfläche doppelt an. Sie gelangen damit in die *Click*-Ereignisprozedur. Geben Sie dort die folgenden Befehle ein:

```
Auszahlungsbetrag = text1.Text
If Auszahlungsbetrag <= Guthaben And Auszahlungsbetrag <= 400 Then
    Guthaben = Guthaben - Auszahlungsbetrag
    If Guthaben > 50 Then
        Guthaben = Guthaben - 50
        Schummelkonto = Schummelkonto + 50
    End If
    MsgBox "Sie erhalten " & Auszahlungsbetrag & " - Neues Guthaben: " &
        Guthaben + Schummelkonto
Else
    MsgBox "Auszahl leider nicht möglich - Kontostand: " &
        Guthaben
End If
```

**9** Starten Sie das Programm durch Drücken der **[F5]**-Taste.

Geben Sie nun in das Textfeld einen Betrag ein, z.B. 300, und klicken Sie auf die Schaltfläche (Sie müssen dazu den alten Inhalt des Textfelds, *Text1*, erst löschen). Sie erhalten eine Meldung über das neue Guthaben. Wiederholen Sie die Auszahlungen. Irgendwann werden Sie feststellen, dass keine Auszahlung mehr möglich ist, obwohl das Guthaben noch ausreichen sollte. Sie erhalten eine entsprechende Meldung, in der auch das wahre Guthaben angezeigt wird. Der Schreck dürfte nicht allzu groß sein, denn es geht ja nicht um echtes Geld, sondern nur um ein kleines Beispiel, das zeigen soll, dass Programme keine Skrupel haben zu »lügen«. Keine Sorge, in Wirklichkeit kommt so etwas natürlich nicht vor.

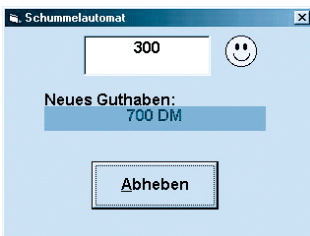


Abbildung 3.8: Darf man dem Programm trauen? Das angezeigte Guthaben stimmt jedenfalls nicht.

Abbildung 3.8 zeigt eine etwas »geschönte« Variante, bei der z.B. das aktuelle Scheinguthaben in einem Bezeichnungsfeld angezeigt wird (das Beispiel finden Sie auf der Buch-CD).

## Programmschleifen

Eine Schleife hat in einer Programmiersprache nichts mit Schuhen zu tun und dient auch nicht dazu, etwas hübsch einzupacken. Eine Schleife ist in diesem Zusammenhang ein anderes Wort für einen Wiederholungsbefehl. Wiederholungen sind eine Spezialität einer jeden Programmiersprache. Mit ihnen wird es möglich, dass ein und derselbe Befehl oder ein und dieselbe Gruppe von Befehlen eine bestimmte Anzahl mal wiederholt werden. Wie oft wird entweder durch einen Grenzwert, der nicht überschritten werden darf, und eine Schrittweite bestimmt oder durch eine Abbruchbedingung, die bei jedem Schleifendurchlauf entweder zu Beginn oder am Ende geprüft wird. Programmschleifen lernen Sie in allen Details in Kapitel 6 kennen. Hier ein kleines Beispiel für einen `For Next`-Befehl, durch den der Wert einer Variablen bei jedem Durchlauf verdoppelt wird:

```
Dim cuBetrag As Currency
Dim inTage As Integer
For inTage = 1 To 21
    cuBetrag = cuBetrag * 2
Next inTage
Msgbox "Am Ende gibt es: " & cuBetrag
```

In diesem Beispiel erhält die Variable `cuBetrag` den Datentyp `Currency`, der sich aufgrund seiner vier festen Nachkommastellen sehr gut für das Rechnen mit Geldbeträgen eignet (es gibt keine Rundungsungenauigkeiten – da es aber nur vier Nachkommastellen gibt ist er nicht eurotauglich, was ein Grund sein kann, warum er mit der nächsten Version abgeschafft wird. Der Datentyp `Currency`, nicht der Euro).

Hier eine Frage für die etwas erfahreneren Programmierer. Warum besitzt die Variable `cuBetrag` am Ende der Schleife immer noch den Wert 0, obwohl sie bei jedem Durchlauf verdoppelt wird? Ganz einfach, weil jede Variable, die einen Zahlendatentyp besitzt, mit dem Wert 0 beginnt. Und 0 mal 2 ergibt immer 0, egal wie oft es wiederholt wird. Es wurde vergessen, die Variable mit dem Wert 0.01 (wenn sie für einen DM-Betrag stehen soll) zu initialisieren. Der folgende Befehl muss vor dem `For Next`-Befehl ausgeführt werden:

```
cuBetrag = 0.01
```

## Einfache Ausgaben über die MsgBox-Funktion

Visual Basic kennt keine Ausgabebefehle, mit denen sich Texte oder Zahlen auf den Bildschirm ausgeben lassen, da es dafür normalerweise die Steuer-



elemente, wie z.B. das Bezeichnungsfeld, gibt. Es gibt lediglich die MsgBox-Funktion, mit der sich allgemeine Mitteilungen bestehend aus einzelnen Sätzen und Zahlen in einer kleinen Box anzeigen lassen.

```
Msgbox "Hallo, das ist eine wichtige Mitteilung"
```

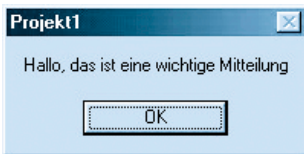


Abbildung 3.9: Für simple Mitteilungen ist die Mitteilungsbox eine einfache Lösung.

Soll die Mitteilungsbox ein kleines Symbol anzeigen, muss dies über eine zweite Zahl festgelegt werden:

```
Msgbox "Ist was?", 32
```



Abbildung 3.10: In der Mitteilungsbox erscheint ein Fragezeichen.

Jetzt erscheint in der Mitteilungsbox ein Fragezeichen. Die Auswahl an Symbolen ist allerdings nicht sehr berauschend:

- 16 – Stop
- 32 – Fragezeichen
- 48 – Ausrufezeichen
- 65 – Informationszeichen

Anstelle der Zahlen ist es üblich, die dafür vorgesehenen Konstanten zu verwenden:

```
Msgbox "Wat is?", vbQuestion
```

Die Konstante `vbQuestion` steht für die Zahl 32 und ist in Visual Basic bereits definiert, sodass sie nicht über den `Const`-Befehl definiert werden muss.

Als dritte Möglichkeit lässt sich der Text festlegen, der in der Überschrift der Mitteilungsbox angezeigt wird:

```
Msgbox "Morgen wird alles besser", vbExclamation, _  
"Prognose der Woche"
```

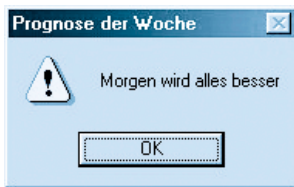


Abbildung 3.11: In der Mitteilungsbox erscheint eine Überschrift.

Weitere Möglichkeiten das Aussehen einer Mitteilungsbox zu ändern gibt es nicht. Wer schöne Mitteilungsboxen gestalten will, muss dazu auf Formulare zurückgreifen, die über das Menükommando PROJEKT/FORMULAR HINZUFÜGEN jederzeit zu einem Projekt hinzugefügt werden können.

## Einfache Eingaben über die Inputbox-Funktion

Als Pendant zur MsgBox-Funktion gibt es bei Visual Basic die InputBox-Funktion. Ihre Aufgabe ist es, eine kleine Box anzuzeigen, in die der Anwender einen beliebigen Text eingeben kann. Der eingegebene Text wird nach dem Bestätigen der Eingabe mit OK einer Variablen zugewiesen und kann anschließend beliebig im Programm weiter verarbeitet werden. Der folgende Befehl fragt den Namen ab:

```
Dim sName As String  
sName = Inputbox("Wie heißt Du denn?")  
Msgbox sName & " ist aber ein sehr schöner Name!"
```

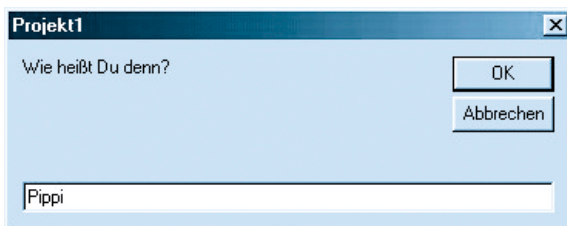


Abbildung 3.12: Die Eingabebox fragt einen Namen ab.

Soll die Überschrift der Eingabebox festgelegt werden, wird diese als zweiter Parameter festgelegt:

```
Dim sName As String  
sName = Inputbox("Wie heißt Du denn?", "Namenscheck")
```

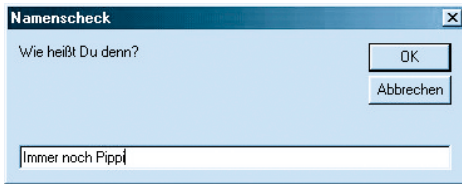


Abbildung 3.13: Die Eingabebox enthält dieses Mal eine Überschrift.

Als kleinen Eingabekomfort gibt es die Möglichkeit, einen Wert vorzugeben, der nur bestätigt werden muss. Dieser sog. *Default-Wert* wird als dritter Parameter beim Aufruf übergeben:

```
Dim sName As String  
sName = Inputbox("Wie heißt Du denn?", "Namenscheck", "Max")
```

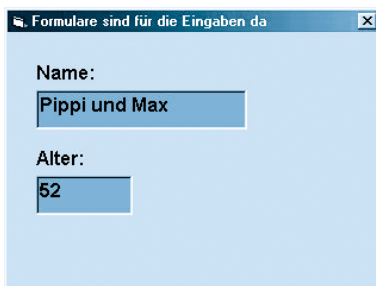


Abbildung 3.14: Die Eingabebox zeigt einen Default-Wert an.

Weitere Gestaltungsmöglichkeiten gibt es auch bei der `Inputbox`-Funktion nicht (immerhin lässt sich die Position auf dem Bildschirm festlegen, an der die Box angezeigt wird). Das ist auch nicht weiter tragisch, denn für Eingaben mit Komfort und allen Schikanen sind bei Visual Basic die Formulare mit ihren Steuerelementen zuständig.

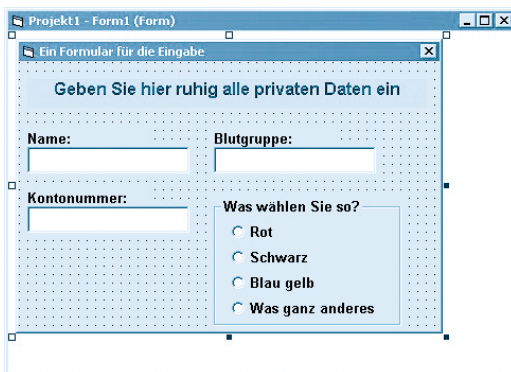


Abbildung 3.15: Ein Formular ist auch für eine einfache Eingabe die bessere Alternative.

## Nützliche Funktionen

Bei Visual Basic muss streng genommen zwischen Befehlen und Funktionen unterschieden werden. Neben den etwas über 90 Befehlen gibt es noch einmal über 150 Funktionen. Sie übernehmen sehr spezielle Dinge, wie z.B. das Zerlegen eines Strings in seine Bestandteile oder das Durchführen einer einfachen mathematischen Berechnung, beispielsweise das Berechnen eines Sinuswertes aus einem Winkel, der in Bogenmaß angegeben wurde, oder das Ziehen einer Quadratwurzel. Es gibt auch viele so genannte *Servicefunktionen*, die dem Programm die aktuelle Uhrzeit verraten, feststellen, ob ein übergebener Wert eine Zahl ist (die `IsNumeric`-Funktion haben Sie bereits in Kapitel 2 kennen gelernt) oder einem Wert einen anderen Datentyp verleihen. Es gibt sogar finanzmathematische Funktionen, mit denen sich etwa die Laufzeit eines Kredits berechnen lässt. Diese Funktionen sind in der Visual Basic-Hilfe ausführlich beschrieben, meistens auch mit Beispielen. Tabelle 3.2 fasst die interessantesten Funktionen in einer Art »Hitparade« zusammen. Die Auswahl ist natürlich subjektiv, doch sind dies Funktionen, die Sie als erfahrener Visual Basic-Programmierer kennen sollten.

Funktion	Was macht sie denn Schönes?
<code>DateAdd</code>	Addiert oder subtrahiert eine Zeiteinheit von einem Datum.
<code>DateDiff</code>	Gibt die Differenz in einer gewünschten Zeiteinheit zwischen zwei Datumsangaben an.
<code>Dir</code>	Gibt den Namen der ersten Datei im angegebenen Verzeichnis zurück, die dem übergebenen Suchbegriff bestehend aus dem Dateinamen und/oder den Platzhaltern * und ? entspricht.
<code>FileLen</code>	Gibt die Größe einer Datei zurück.
<code>FormatCurrency</code>	Gibt eine Zeichenkette zurück, die einen Geldbetrag mit dem über die Systemsteuerung eingestellten Währungssymbol und zwei Nachkommastellen enthält.
<code>FormatDateTime</code>	Gibt eine Zeichenkette zurück, die ein Datum und/oder eine Uhrzeit in einem festgelegten Format enthält.
<code>FormatNumber</code>	Gibt eine Zeichenkette zurück, die eine Zahl mit einer festgelegten Anzahl an Nachkommastellen enthält.

Funktion	Was macht sie denn Schönes?
Hex	Gibt den Hexadezimalwert einer Zahl als Zeichenkette zurück.
Int	Schneidet den Nachkommaanteil einer Zahl ab und gibt nur den ganzzahligen Wert zurück.
IsNumeric	Prüft, ob sich eine Zeichenkette in eine Zahl umwandeln lässt.
Log	Berechnet den Logarithmus zur Basis 10.
Now	Steht für die aktuelle Systemzeit.
Rnd	Liefert eine »Zufallszahl« zwischen 0 und 1.
Round	Rundet eine Zahl auf eine beim Aufruf angegebene Genauigkeit an Ziffern.
Sqr	Berechnet die Quadratwurzel einer Zahl.

Tabelle 3.2: Nützliche Visual Basic-Funktionen

## Befehl, Funktion – gibt es einen Unterschied?

Was unterscheidet eine Funktion von einem Befehl – beide erledigen doch eine Aufgabe? Die Antwort setzt bereits die etwas fortgeschrittenere Programmierung voraus. Funktionen dürfen nur auf der rechten Seite einer Zuweisung aufgeführt werden. Hier ein Beispiel für die Wurzelfunktion Sqr:

```
Sqr(10)
```

Der Funktionsaufruf ist zwar korrekt, aber noch nicht vollständig, denn irgendetwas muss mit dem Ergebnis der Funktionen (Programmierer nennen das den *Rückgabewert*) schließlich geschehen. Er kann einer Variablen zugewiesen werden:

```
w = Sqr(10)
```

Jetzt ist der Befehl vollständig. Wird er ausgeführt, erhält die Variable *w* das Ergebnis des Funktionsaufrufs. Damit wird sicherlich etwas klarer, was vorhin mit der etwas ominösen Behauptung gemeint war, dass Funktionen immer auf der rechten Seite einer Zuweisung stehen müssen. Der folgende Befehl ist nicht erlaubt:

```
Sqr(10) = w
```

Bei Befehlen gibt es keinen Rückgabewert. Sie stehen immer links vom Gleichheitszeichen, sofern ein solches überhaupt involviert ist. Ein Befehl,

der einer Funktion am nächsten kommt, ist der Date-Befehl, der über das Systemdatum geändert wird:

```
Date = "25.10.63"
```

Der Befehl steht links vom Gleichheitszeichen. Es spricht nichts dagegen, einen Befehl mit einer Funktion zu kombinieren, die dann ihren rechtmäßig angestammten Platz auf der rechten Seite vom Gleichheitszeichen einnehmen muss:

```
Date = DateAdd("d", Date, -1)
```

Dieser Befehl setzt das aktuelle Datum um einen Tag zurück, wobei mit Date (das ist eine Funktion – sie steht für das aktuelle Datum) eine zweite Funktion involviert ist. Dass es sowohl einen Date-Befehl als auch eine Date-Funktion gibt, macht Visual Basic nichts aus, denn es kann beide aufgrund einer einfachen Regel auseinanderhalten. Wissen Sie noch wie diese Regel lautet? Richtig, eine Funktion darf nie links vom Gleichheitszeichen aufgeführt werden.

## Der Umgang mit Projekten

Zum Abschluss dieses eher theoretischen Kapitels noch ein wenig Praxis. Zu den Dingen, die ein(e) Visual Basic-Programmierer(in) beherrschen muss, gehört der Umgang mit Projekten. Sie wissen bereits, dass Visual Basic-Programme immer in Projekten gespeichert werden. Ein Projekt umfasst alle Formulare, Module, Klassenmodule usw., die zu dem Programm gehören und die im Laufe der Zeit in das Projekt geladen wurden. Der Inhalt eines Projekts wird im Projekt-Explorer-Fenster angezeigt. Zu den wichtigsten Aktionen im Zusammenhang mit den Projekten gehören:

- Das Anlegen eines neuen Projekts
- Das Öffnen eines bereits vorhandenen Projekts
- Das Speichern eines Projekts

## Das Anlegen eines neuen Projekts

Diesen Schritt haben Sie bereits in Kapitel 2 kennen gelernt. Ein neues Projekt wird entweder mit dem Start von Visual Basic durchgeführt, Sie können es aber auch jederzeit nachträglich erledigen. Öffnen Sie dazu das DATEI-Menü und wählen Sie den Eintrag NEUES PROJEKT (einfacher geht es meistens mit **[Strg] + [N]**). Visual Basic präsentiert Ihnen den üblichen Auswahldialog, aus dem Sie den anzulegenden Projekttyp auswählen sollen (dieser Aus-

wahldialog erscheint aber nicht immer – in diesem Fall wird ein STANDARD-EXE-Projekt angelegt). Die einzelnen Projekte unterscheiden sich übrigens nur durch Details, wie z.B. die anfänglich geladenen Module oder bestimmte Projekteigenschaften, die alle nachträglich geändert werden können. In diesem Sinne hat die Auswahl eines Projekttyps keine allzu große Bedeutung.



Abbildung 3.16: Über das DATEI-Menü wird ein neues Projekt angelegt.

## Das Öffnen eines bereits vorhandenen Projekts

Möchten Sie ein Projekt erneut laden, das bereits abgespeichert wurde, müssen Sie die Projektdatei (zu erkennen an der Erweiterung `.Vbp`) öffnen. Öffnen Sie dazu das DATEI-Menü und wählen Sie den Eintrag PROJEKT ÖFFNEN (einfacher geht es meistens mit `[Strg] + [O]`). Sollte ein Projekt bereits geladen sein und sollten an dem Projekt Änderungen vorgenommen worden sein, erhalten Sie die Gelegenheit das Projekt zu speichern (mehr dazu gleich). Visual Basic bietet Ihnen einen speziellen Auswahldialog mit zwei Möglichkeiten an:

- Auswahl der Projektdatei aus einem beliebigen Verzeichnis (Registerkarte VORHANDEN).
- Auswahl der zuletzt gespeicherten Projekte (Registerkarte AKTUELL).

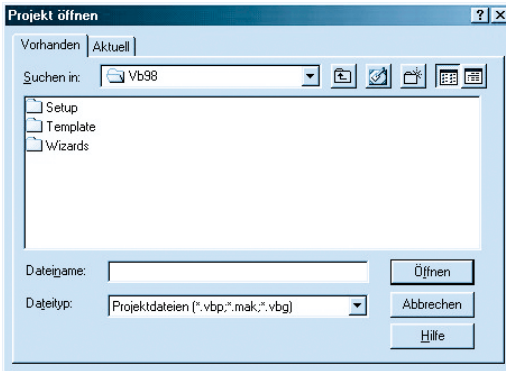


Abbildung 3.17: Aus diesem Dialogfeld wird ein bereits gespeichertes Projekt zum Öffnen ausgewählt.

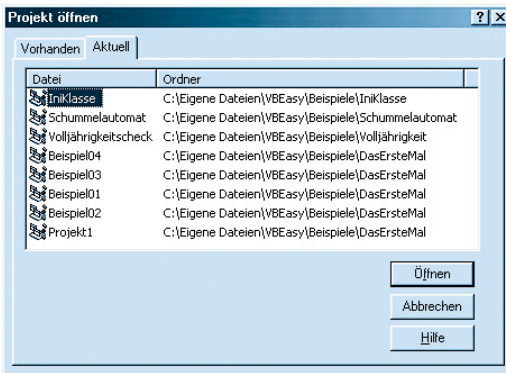


Abbildung 3.18: In der Registerkarte AKTUELL stehen die zuletzt gespeicherten Projekte zum Abruf bereit.

Ein wenig ungünstig ist, dass in der Registerkarte VORHANDEN immer das Visual Basic-Programmverzeichnis voreingestellt wird. Hier sollten eigentlich keine Visual Basic-Projekte gespeichert werden.

Als dritte Möglichkeit ein Projekt zu öffnen bietet das DATEI-Menü am Ende die vier zuletzt gespeicherten Projekte an. Dies ist oft der schnellste Weg, um ein kürzlich bearbeitetes Projekt erneut zu öffnen.

## Das Speichern eines Projekts

Bevor Sie Visual Basic beenden, müssen Sie das Projekt speichern, denn ansonsten sind alle an dem Projekt seit dem letzten Speichern vorgenommen Änderungen verloren. Wenn Sie die in Kapitel 1 vorgeschlagene Einstellungsänderung in den OPTIONEN vorgenommen haben, sollte eine Abfrage



für das Speichern der seit dem letzten Speichern geänderten Module vor jedem Programmstart erfolgen, sodass das Speichern, bevor Sie Visual Basic beenden, nicht immer notwendig ist.

Visual Basic zeigt Ihnen vor dem Beenden eines Projekts alle Module an, in denen seit dem letzten Speichern Änderungen vorgenommen wurden. Sie können die Auswahl komplett mit *OK* bestätigen oder einzelne Module abwählen, die nicht gespeichert werden sollen.

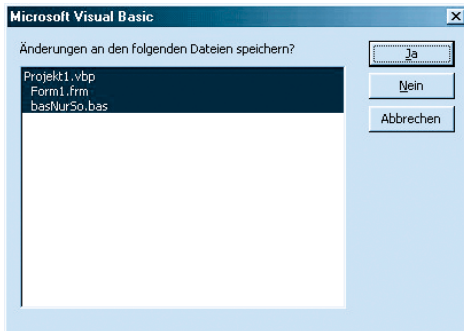


Abbildung 3.19: Vor dem Beenden eines Projekts werden die zu speichernden Module angezeigt.

### Tipp

Über das Menükommando DATEI/PROJEKT SPEICHERN (oder am schnellsten über das Diskettensymbol in der Symbolleiste) wird das komplette Projekt, d.h. alle geänderten Module, gespeichert. Das ist der schnellste Weg, um ein Projekt zu speichern. Wenn Sie die von Word bekannte Tastenkombination **Strg** + **S** instinktiv verwenden, speichern Sie damit nur das aktuelle Modul.

## Ausblick – Wie geht es weiter?

Dies war das wichtigste Kapitel des Buchs. Sie haben in diesem Kapitel das ABC der Programmierung mit Visual Basic, d.h. die wichtigsten Befehle der Programmiersprache, kennen gelernt. Nebenbei haben Sie gelernt, wie ein Computer »denkt«, indem Sie das Prinzip von Entscheidungen und Programmschleifen kennen gelernt haben. Diese Dinge sind bei 90% aller Programmiersprachen nahezu gleich. Sie müssen nicht alles auf Anhieb verstanden haben, um fortfahren zu können. Sie werden aber alle in diesem

Kapitel vorgestellten Befehle für fast alle Ihre künftigen Programme benötigen, sodass Sie dieses Kapitel noch öfter durchblättern und einzelne Passagen wiederholen werden.

Im nächsten Kapitel wird es sehr praktisch. Sie lernen mit den Steuerelementen eine Vielzahl unterschiedlicher Bausteine kennen, die in der Werkzeugammlung (einem Fenster der Visual Basic-Entwicklungsumgebung) angeboten und auf einem Formular angeordnet werden, und dort verschiedene Möglichkeiten für die Eingabe von Texten und Zahlen oder für die Auswahl von Optionen oder Namen aus einer Liste. Der Umgang mit Steuerelementen ist sehr einfach. Sie sind ein Grund dafür, dass sich Visual Basic so schnell einer so großen Beliebtheit erfreut hat.