1. Objekte

1.1 Was ist ein Objekt?

Praktisch alles, was auf dem Bildschirm eines Computers, der unter dem Betriebssystem Windows läuft, ist ein Objekt: der Mauszeiger, der Bildschirmhintergrund, das Fenster eines gerade geöffneten Programmes, ein Eintrag in der Menüzeile, ein Eingabefeld, eine Schaltfläche usw.

Gerade am Beispiel eines Programm-Fensters wird deutlich, dass ein Objekt weitere Objekte enthalten kann, wobei z.B. ein Menüeintrag nicht ohne das übergeordnete Objekt des zugehörigen Fensters existieren kann. Objekte, die andere Objekte enthalten, bezeichnet man deshalb auch als Container.

Jedes Objekt besitzt bestimmte Eigenschaften, die z.B. ihr Erscheinungsbild bestimmen, außerdem kann es in der Regel auch auf verschiedenste Ereignisse reagieren. Damit man jedes Objekt nicht von Grund auf neu definieren muss, greift die objektorientierte Programmierung auf Bibliotheken (sog. Klassen-Bibliotheken) zurück, in der Objekte enthalten sind, die dem Benutzer zur Verfügung stehen. Die Definition eines solchen Objektes nennt man Klasse, das heißt, gleiche Objekt gehören zu der gleichen Klasse.

1.2 Namen, Eigenschaften, Ereignisse und Methoden

Jedes Objekt trägt einen eindeutigen **Namen**, über den auf das Objekt zugegriffen werden kann. Ist das Objekt Bestandteil eines Containers, ist ein Zugriff auf das Objekt „von außen“ nur über den Namen des zugehörigen Containers möglich (sofern der Programmierer diesen Zugriff erlaubt hat).

Außerdem verfügt jedes Objekt über unterschiedliche **Eigenschaften**, dazu gehören z.B. die Koordinaten auf dem Bildschirm, die verwendete Schriftart, die Vorder- oder Hintergrundfarbe usw.

Die meisten Objekte sind in der Lage, auf bestimmte Ereignisse zu reagieren, z.B. einen Tastendruck oder einen Mausklick. Die mit diesen **Ereignissen** verknüpften Programmabläufe nennt man **Methoden**.

1.3 Zugriff auf ein Objekt

Um die Eigenschaften oder Methoden eines Objektes zu verwenden, wird eine im Prinzip recht einfache Schreibweise verwendet. Sie erfolgt grundsätzlich nach dem Muster

```
name.eigenschaft        oder        name.methode
```

Ist das Objekt Bestandteil eines anderen Objektes (Container) und hat der Programmierer einen Zugriff erlaubt, so erfolgt der Zugriff z.B. über

```
name1.name2.eigenschaft     bzw.     name1.name2.methode
```

wobei `name1` der Name des (Container-) Objektes und `name2` der Name des untergeordneten Objektes ist.

1.4 Beispiel

Unter Windows stellt das Programm Excel eine `Application` dar. Alle darin geöffneten Dateien stellen die `Workbooks` dar, dabei ist die geöffnete Datei (z.B. „Step1.xls“) als erste (und ggf. einzige) Datei der Eintrag `Workbooks(1)`. Jede Excel-Datei kann mehrere Tabellen enthalten, die als `Worksheets` bezeichnet werden. Die erste Tabelle von Step1.xls ist dabei `Worksheets(1)`. Die Befehlsschaltfläche „Anzeigen“ sei das erste Objekt der ersten Tabelle von Step1.xls, welches unter den `Shapes` aufgeführt wird. Dieses Objekt hat als `Shapes(1)` eine Eigenschaft Name, auf den nun wie folgt zugegriffen werden kann:

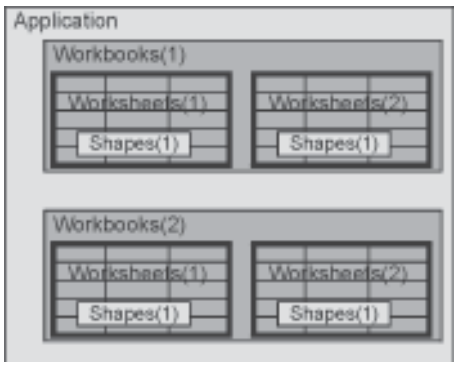

Abb. 1: Objekt-Hierarchie unter Excel

```
Application.Workbooks(1).Worksheets(1).Shapes(1).Name
```

Statt des vollständigen Objektpfades kann in der Regel auch eine verkürzte Schreibweise verwendet werden, wenn innerhalb eines Objektes auf ein untergeordnetes Objekt zugegriffen werden soll, z.B. durch

```
ActiveSheet.Shapes(1).Name
```

2. Arbeitsbereich vorbereiten

2.1 Steuerelement-Toolbox

Excel „merkt“ sich die letzte Einstellung der während der Arbeit zur Verfügung stehenden Symbolleisten. Um bestimmte Objekte auf dem Tabellenblatt unterzubringen, wird die Steuerelement-Toolbox benötigt. Wird diese nicht angezeigt, muss sie aktiviert werden über das Menü

Ansicht - Symbolleisten - Steuerelement-Toolbox

auf gleichem Wege kann sie auch wieder aus der Arbeitsfläche ausgeblendet werden, wenn sie nicht mehr benötigt wird.

2.2 Darstellungsoptionen

Abb. 1: Toolbox als Leiste in der Menüzeile

Diese Toolbox kann als eigenes Fenster auf der Arbeitsfläche erscheinen, das sich an beliebige Positionen verschieben lässt, es kann aber auch in die Schaltflächenleiste unter der Menüzeile verschoben werden. Wenn sie als Fenster erscheint und man sie in der Menüzeile platzieren möchte, klickt man das Fenster im oberen Balken an und schiebt es mit gedrückter Maustaste in einen freien Bereich der Menüzeile. Um die Toolbox wieder als Fenster auf der Arbeitsfläche zu erhalten, klickt man stattdessen die Doppellinie am linken Rahmenrand (links neben dem Bearbeiten-Symbol) an und kann sie nun genauso auch wieder auf die Arbeitsfläche ziehen.

2.3 Toolbox-Symbole

Standardmäßig werden auf der Toolbox 3 Steuersymbole, 11 platzierbare Elemente und ein Erweiterungsfeld angezeigt, die im Folgenden näher erläutert werden, in [ ] ist außerdem jeweils die Objektklasse angegeben. Der Standardname, der beim Platzieren eines neuen Objektes von Excel-VBA automatisch vergeben wird, wird aus dem Klassennamen und einer angehängten fortlaufenden Zahl gebildet:

Bearbeiten / Anzeigen:

Umschalten zwischen Bearbeiten und Ausführen von Objekten

Ein- / Ausschalten des Eigenschaften-Fensters, mit dem die Einstellungen des jeweils ausgwählten Objektes bearbeitet werden können

Aufrufen des Visual-Basic-Code-Editors (erfolgt im Modus Bearbeiten ebenfalls bei einem Doppelklick auf das ausgewählte Objekt, dann wird allerdings ggf. automatisch eine neue Prozedur erstellt oder, falls bereits vorhanden, angezeigt.)

Platzierbare Objekte

Kontrollkästchen, kann drei Zustände annehmen: 0 = deaktiviert, 1 = aktiviert, 2 = unbestimmt. [CheckBox]

Textfeld, kann beliebige Eingaben aufnehmen, die über die Tastatur eingegeben werden können. [TextBox]

Befehlsschaltfläche, nach einem Mausklick auf diesen Button wird ein Ereignis ausgelöst. [CommandButton]

Optionsfeld, sinnvoll nur in einer Gruppe mit weiteren Optionsfeldern einzusetzen. Erlaubt die Auswahl von genau einer aus mehreren Möglichkeiten, das jeweils aktivierte Feld hat den Zustand True, alle anderen Felder sind False. (Wird häufig innerhalb von Frames eingesetzt.) [OptionButton]

Listenfeld (mehrzeiliges Textfeld), erlaubt die Auswahl eines Eintrags von mehreren aus einer vordefinierten Liste von Einträgen, kann mehrzeilig angezeigt werden.
[ListBox]

Kombinationsfeld, wie Listenfeld, wird einzeilig angezeigt und erlaubt durch einen Button das „Aufklappen“ eines Listenfeldes, aus dem dann ein Eintrag ausgewählt werden kann.
[ComboBox]

Umschaltfläche, wie Kontrollkästchen, kann aber nur zwei Zustände (True, False) annehmen und ändert je nach Zustand seine Form (erhaben / vertieft).
[ToggleButton]

Drehfeld, besteht aus zwei mit Pfeilen versehenen Schaltflächen, mit denen ein Zahlenwert eingestellt werden kann, der zwischen einer vorgebbaren Unter-/ Obergrenze liegt. Die Orientierung kann auf horizontale oder vertikale Anordnung eingestellt werden.
[SpinButton]

Bildlaufleiste, Verwendung wie Drehfeld, jedoch mit einem zusätzlichen Balken zwischen den Schaltflächen, der eine schnellere Einstellung des gewünschten Zahlenwertes erlaubt.
[ScrollBar]

Bezeichnung, erzeugt ein Textfeld mit statischem Text, der im Modus Ausführen nicht verändert werden kann.
[Label]

Bild, erzeugt ein Feld, das zur Darstellung von Bitmapgraphiken (vom Typ .BMP) geeignet ist.
[Image]

Weitere Elemente:

Weitere Steuerelemente, öffnet ein Dialogfeld mit einer großen Auswahl weiterer platzierbarer Objekte. Deren Verwendung setzt allerdings eine entsprechende Literatur voraus, um die damit verknüpften Eigenschaften und Methoden anwenden zu können, da hier in der Regel keine automatischen Hilfe-Informationen zur Verfügung stehen.

3. Anlegen einer Schaltfläche auf einem Tabellenblatt

Damit aus einer Excel-Tabelle heraus ein eigenes Programm oder Formular verwendet werden kann, muss dieses (nachdem es angelegt worden ist), auch aktiviert werden können. Dazu ist es am einfachsten, eine Schaltfläche einzubauen, die auf die Methode „Maus-Klick“ reagiert und daraufhin eine entsprechende Anweisung ausführt.

3.1 Erzeugen einer Schaltfläche zum Anzeigen eines Formulars

In der Toolbox kontrolliert man, ob das Symbol für den Bearbeitungsmodus aktiviert ist (es ist dann hell unterlegt) und klickt dann auf das Symbol für eine Befehlsschaltfläche, das dann vertieft dargestellt wird. Bewegt man nun den Mauszeiger auf das Tabellenblatt, wird als Mauszeiger statt des üblichen Pfeils ein Kreuz angezeigt. Anschließend zieht man an der gewünschten Position auf dem Tabellenblatt mit gedrückter linker Maustaste ein Rechteck von ungefähr der gewünschten Größe. Nach dem Loslassen der Maustaste wird ein graues Rechteck mit der Beschriftung CommandButton1 angezeigt, das von mehreren kleinen Quadraten umgeben ist, die das Objekt als markiert kennzeichnen und mit denen die Größe der Schaltfläche verändert werden kann.

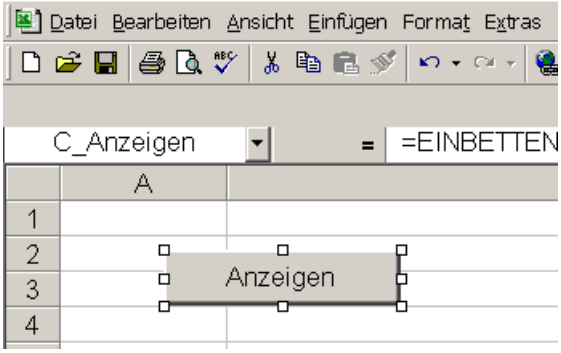

Abb. 2: Die fertige Schaltfläche „Anzeigen“

3.2 Bearbeiten der Eigenschaften der Schaltfläche

Um die Eigenschaften der eben angelegten Schaltfläche bearbeiten zu können, benötigt man das Eigenschaftsfenster. Dieses erhält man auf zwei möglichen Wegen:

- Die noch markierte neue Schaltfläche wird mit der rechten Maustaste angeklickt und in dem dabei aufklappenden Menü der Eintrag „Eigenschaften“ angewählt
- In der Toolbox (s. S. 2) klickt man auf das Symbol „Ein-/Ausschalten des Eigenschaftsfensters“

In beiden Fällen wird das in Abb. 3 angezeigte Fenster eingeblendet, das beliebig auf dem Bildschirm platziert werden kann.

Die bearbeitbaren Eigenschaften werden entweder alphabetisch oder nach Kategorien angezeigt. Welche der beiden Darstellungen man wählt, ist Geschmacks- (und Gewohnheits-) -sache, hat aber sonst keine weitere Bedeutung.

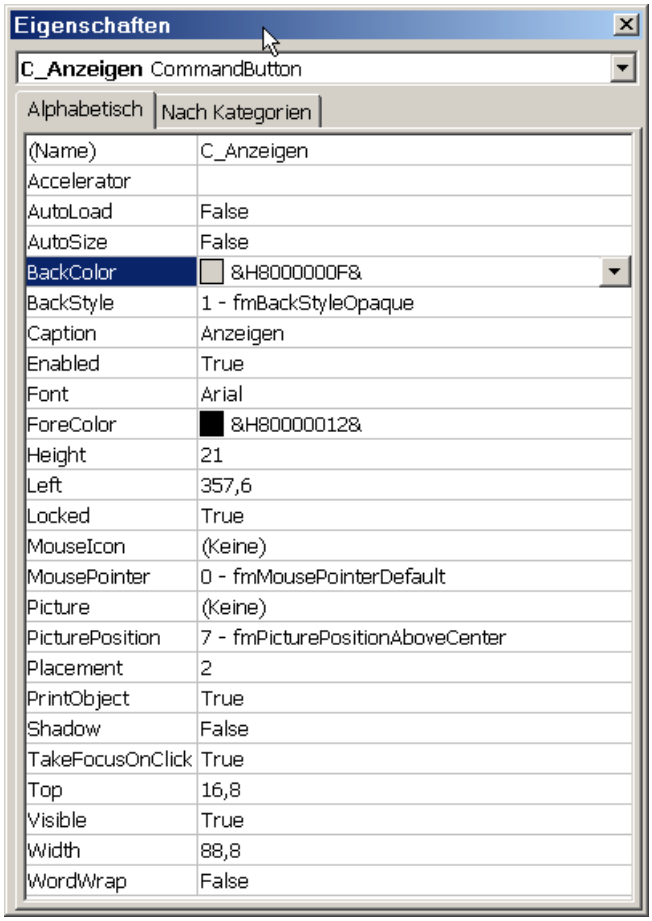

Abb. 3: Eigenschaftsfenster auf dem Tabellenblatt

3.2.1 (Name) - Objektnamen festlegen

Die wichtigste Eigenschaft ist der Objektname, denn über diesen wird künftig auf alle Ereignisse und Eigenschaften dieses Objektes zugegriffen. Wird dieser Name später verändert, müssen in allen Programmen die Bezüge auf dieses Objekt ebenfalls geändert werden, da diese nicht automatisch angepasst werden! Deshalb sollte diese Bezeichnung sinnvoll und eindeutig gewählt werden. Hier wird als Beispiel der Name „C_Anzeigen“ verwendet.

Wichtig: Der hier gewählte Name darf nur aus Buchstaben und Zahlen sowie dem Unterstrich _ gebildet werden und keine Leerzeichen oder andere Sonderzeichen enthalten.

Zur besseren Übersichtlichkeit bewährt es sich, den Namen so zu wählen, dass die zugehörige Objektklasse daraus entnommen werden kann, wobei die Verwendung von Groß- und Kleinschreibung die Lesbarkeit erleichtert. Es gibt dafür keine verbindlich vorgeschriebenen Regeln, die folgenden Beispiele können aber als Anregung dienen:

Objektklasse	Beispiel-Funktion	kleine Projekte	größere Projekte
CheckBox	Bearbeitung erlauben	Cx_Allowedit	ChbAllowEdit
TextBox	Eingabefeld für Dateinamen	T_DatNam	TxtDatNamOpen
CommandButton	Anwendung schließen	C_Close	CmdCloseAll
OptionButton	Datei Lesen / Schreiben	O_ReadWrite	OptReadWrite
ListBox	Dateityp auswählen	L_DatTyp	LbxDatTyp
ComboBox	Datei auswählen	Cx_File	CbxFileName
ToggleButton	Groß-/Kleinschreibung beachten	Tb_Upcase	TbtCheckUpCase
SpinButton	Farbe einstellen	Sb_FCol	SbtForeCol
ScrollBar	Lautstärke einstellen	S_Loudness	ScbLoudness
Label	Bezeichnungsfeld „Dateiname“	L_DatNam	LabDatNam
Image	Graphische Darstellung	I_BmpGraph	ImgBmpGraphic

Solange kein eigener Name vergeben wurde, bezeichnet VBA die Objekte gemäß ihrer Objektklasse und hängt eine fortlaufende Nummer an, z.B. CheckBox1 für die erste, CheckBox2 für die zweite Checkbox usw.

3.2.2 Caption - Beschriftung festlegen

Einige Objekte werden zusammen mit einer Beschriftung angezeigt, dazu gehören z.B. CommandButton, CheckBox, ToggleButton und OptionButton, andere kommen ohne eine Beschriftung aus. Sofern eine Beschriftung vorgesehen ist, wird im Eigenschaftsfenster der Eintrag Caption angezeigt, dessen Wert beliebig geändert (und sogar gelöscht) werden kann. Der Übersichtlichkeit halber sollten diese Beschriftungen aussagekräftig und nicht zu lang sein, dabei sind alle über die Tastatur eingeb- und darstellbaren Zeichen inklusive Leerzeichen erlaubt. Hier wird als Caption z.B. „Anzeigen“ eingegeben.

Um eine Schaltfläche zu erstellen, können die übrigen Eigenschaften zunächst ignoriert werden . die so angelegte Schaltfläche ist prinzipiell nun funktionsfähig - allerdings fehlt noch die Angabe der zugehörigen Aktion, die später nach einem Anklicken der Schaltfläche erfolgen soll, dazu benötigt man ein entsprechendes Objekt.

4. Anlegen eines eigenen Formulars

Um ein eigenes Formular als anzuzeigendes Objekt zu erzeugen, muss der VBA-Code-Editor gestartet werden, dafür gibt es folgende drei Optionen:

- Anklicken des Toolbox-Symbols „Aufrufen des Visual-Basic-Code-Editors“
- Menüauswahl Extras - Macro - Visual-Basic-Editor
- Tastenkombination <ALT>-<F11>

In jedem Fall öffnet sich eine neue Anwendung mit drei Fenstern, was z.B. etwa wie in Abb. 4 aussehen könnte:

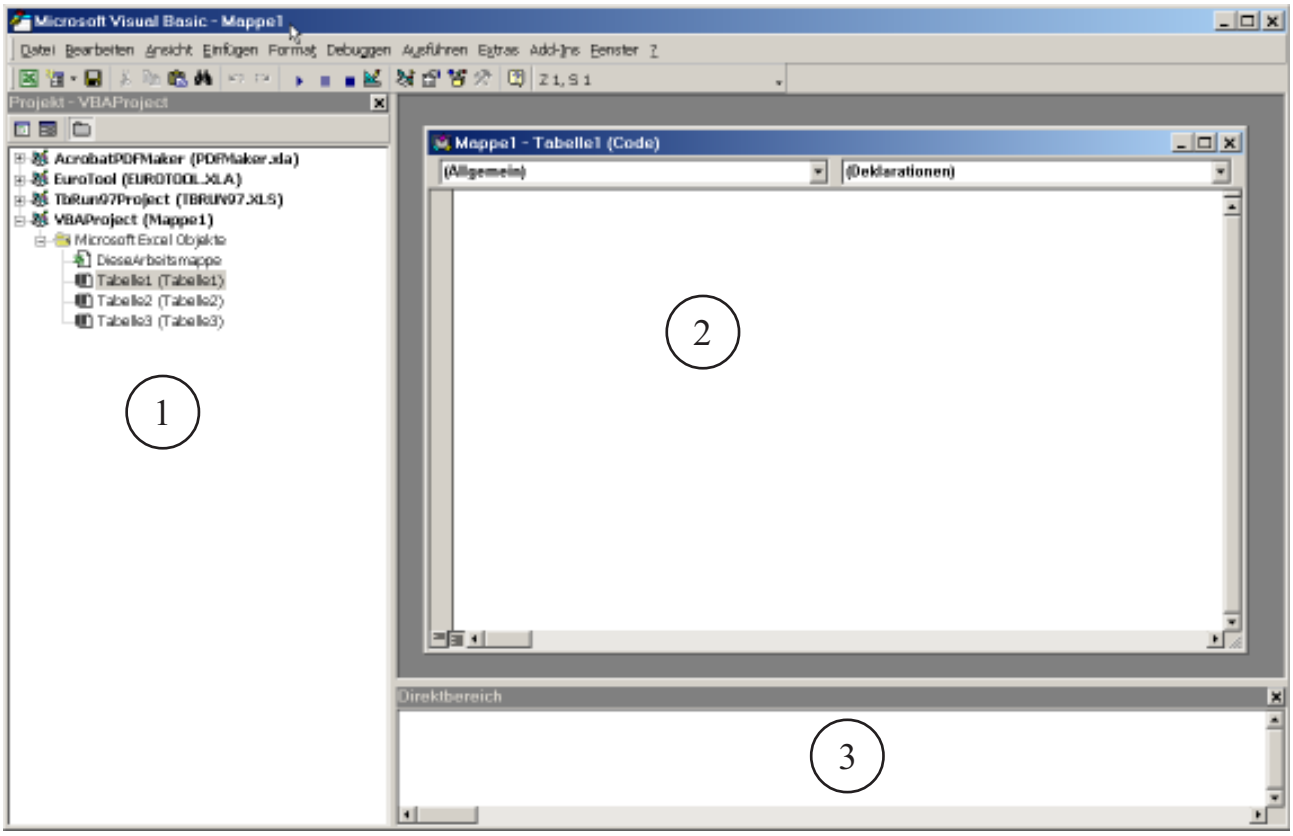

Abb. 4: Arbeitsoberfläche des VBA-Code-Editors bei erstmaligem Öffnen

Fenster 1: „Projekt-Explorer“ - Hier werden alle Workbooks und ihre Worksheets angezeigt, die in der Application Excel derzeit geöffnet sind. Durch einen Klick auf das dem Objektnamen vorangestellte Plus- bzw. Minus-Zeichen können untergeordnete Objekte ein- bzw. ausgeblendet werden.

Fenster 2: „Code-Fenster“ - Hier wird der Code angezeigt, der für Objekte auf der markierten Tabelle vorhanden ist - da noch kein Code eingegeben wurde, ist dieses Fenster noch leer.

Fenster 3: „Debug-Fenster“ - Hier können Variablen ausgegeben oder der Zustand von Objekt-Eigenschaften abgefragt werden.

Durch einen Klick auf die ganz linke Schaltfläche in der Menü-Leiste mit dem typischen Excel-Symbol kann jederzeit auf das Tabellenblatt zurück gewechselt werden..

4.1 Erzeugen eines neuen Formulars

Um ein neues Formular anzulegen, gibt es auch hier wieder 2 Möglichkeiten:

- Menüauswahl Einfügen - Userform
- rechter Mausklick in einem freien Bereich des Fensters 1, Auswahl des Eintrags Einfügen - Userform

In jedem Fall erscheint in dem Projektexplorer nun ein neuer Eintrag „Formulare“, bei dem als untergeordnetes Objekt das neue Formular mit dem vorgegebenen Namen UserForm1 aufgeführt ist. Außerdem werden zwei weitere Fenster angezeigt: Eine Werkzeug-Box, die frei verschoben werden kann, sowie ein neues Fenster im Code-Bereich, in dem eine graue gepunktete Fläche mit dem Titel Userform1 dargestellt wird.

Um die Eigenschaften dieses Formulars bearbeiten zu können, wird nun noch das Eigenschaftsfenster benötigt, das man durch Anklicken des entsprechenden Symbols auf der Menüleiste oder die Funktionstaste <F4> eingeblendet erhält. Üblicherweise wird dieses unter dem Projektexplorer (Fenster 1) angezeigt, kann aber auch an eine beliebige andere Stelle auf dem Bildschirm verschoben werden.

4.2 Bearbeiten der Eigenschaften des Formulars

Für die Bearbeitung der Eigenschaften gelten prinzipiell die gleichen Vorgehensweisen wie schon bei der Schaltfläche in Kap. 3.2 beschrieben, so wie dort genügt es zunächst, die beiden Eigenschaften (Name) und Caption anzupassen. Für die weiteren Beispiele wird vorausgesetzt, dass das Formular den Namen Hauptformular und die Bezeichnung „Hauptformular“ erhält.

5 Programmieren einer auszuführenden Aktion

Zunächst soll lediglich erreicht werden, dass nach einem Klick auf die Schaltfläche im Tabellenblatt, die im Kap. 3 angelegt wurde, das in Kap. 4 erzeugte Formular angezeigt wird. Auch hier gibt es wieder zwei Möglichkeiten:

- Man klickt auf die linke Auswahlleiste des Code-Fensters von Tabelle 1, das zu dieser Tabelle gehört und wählt daraus den Eintrag „C_Anzeigen“ aus. Dabei wird automatisch ein Prozedurrumpf erzeugt, der nur noch durch die entsprechenden Anweisungen ergänzt werden muss.
- Man wechselt in die Tabellenansicht von Excel und macht einen Doppelklick auf die dort bereits erzeugte Schaltfläche „Anzeigen“. Auch hier wird dann automatisch der VBA-Editor aufgerufen und ein passender Prozedurrumpf erzeugt.

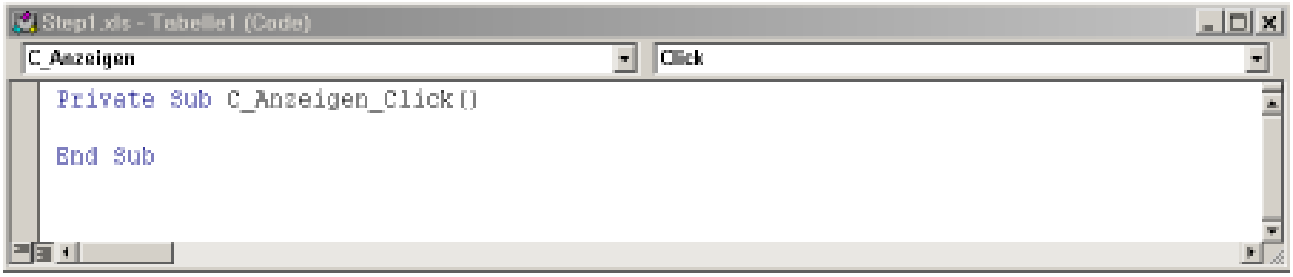

Abb. 5: Automatisch angelegter Prozedurrumpf für die Schaltfläche „C_Anzeigen“

Auch später wird es häufig nötig sein, zwischen Tabellenansicht und VBA-Editor zu wechseln. Aus der Tabellenansicht gibt es dafür mehrere Möglichkeiten:

- Doppelklick auf die markierte Schaltfläche (siehe oben)
- Anklicken des Toolbox-Symbols „Aufrufen des Visual-Basic-Code-Editors“
- Menüauswahl Extras - Macro - Visual-Basic-Editor
- Tastenkombination <ALT>-<F11>

In jedem Fall öffnet sich der VBA-Editor, wobei allerdings nur im ersten Beispiel automatisch auch schon ein passender Prozedurrumpf bereitgestellt wird, in dem man nur noch eine Anweisung einzutragen braucht.

Man beachte die beiden Textfelder am oberen Rand des Fensters in Abb. 5: das linke Textfeld enthält den Namen des Objektes „C_Anzeigen“, das rechte Textfeld die (automatisch vorgegebene) Methode „Click“, was bedeutet, dass hier ein einfacher Klick mit der linken Maustaste auf die Schaltfläche bearbeitet werden wird.

5.1 Reaktion der Schaltfläche C_Anzeigen auf das Ereignis „Click“

Die mit dem Mausereignis verbundene Aktion soll bewirken, dass ein bestimmtes Formular angezeigt wird, in unserem Fall das Formular Hauptformular.

Um den zugehörigen Code einzugeben, klickt man in die leere Zeile unter Private und schreibt nach einem Tabulator-Zeichen (das Einrücken der 2. Zeile mit der Tabulatortaste erhöht dabei die spätere Lesbarkeit der Programme) den Namen des Formulars hin, gefolgt von einem Punkt. Da das Formular mit dem angegebenen Namen bereits existiert, klappt automatisch eine Auswahlliste auf, die bei weiteren Eingaben auf die vorhandenen möglichen Eigenschaften oder Methoden springt. Hier ist die Methode „Show“ gefragt, denn das Formular soll angezeigt werden.

Damit ist die Prozedur bereits fertig:

```
Private Sub C_Anzeigen_Click()
    Hauptformular.Show
End Sub
```

Dass es sich hier um eine Prozedur handelt, erkennt man an dem Schlüsselwort Sub, während das Schlüsselwort Private angibt, dass diese Prozedur nur von dem unmittelbar übergeordneten Objekt aufgerufen werden darf, was in diesem Fall das Tabellenblatt (ActiveSheet) ist.

5.2 Testen der Funktion der Schaltfläche C_Anzeigen

Wie schon so oft gibt es auch hier wieder zwei Möglichkeiten:

- Wechseln in das Tabellenblatt, Bearbeitungsmodus beenden, Mausklick auf die Schaltfläche C_Anzeigen
- Klick auf das kleine blaue, nach rechts weisende Dreieck in der Menüzeile des VBA-Editors (dann wird der Code für das aktuelle Objekt ausgeführt, dessen Code-Fenster gerade aktiviert ist).

In beiden Fällen sollte das Formular Hauptformular auf dem Tabellenblatt angezeigt werden. Um es zu schließen, muss man allerdings noch auf das Kreuz am oberen rechten Fesnterrand des Formulars klicken, da derzeit ja noch keine andere Aktion unterstützt wird.

5.3 Userform Hauptformular mit Schaltfläche und passender Prozedur versehen

Ähnlich wie im Kap. 2 kann man auch das Formular mit einem CommandButton versehen. Dazu aktiviert man im VBA-Editor das Fenster mit dem Formular in der Objekt-Ansicht (vergl. Kap. 4.1), klickt in der Werkzeugbox das Schaltflächensymbol an und zeichnet auf dem Formular innerhalb des grau gerasterten Feldes ein passendes Rechteck.

Da es Aufgabe dieser Schaltfläche sein soll, das Formular wieder zu schließen, werden die Eigenschaften dieser Schaltfläche wie folgt gesetzt:

```
(Name)      C_Close
Caption     Schließen
```

Nach einem Doppelklick auf die Schaltfläche wird automatisch das Code-Fenster für das Formular eingeblendet und der dazu passende Prozedurrumpf erzeugt. Nach Eintrag der entsprechenden Anweisung erhält man

```
Private Sub C_Close_Click()
    Hauptformular.Hide
End Sub
```

was dazu führt, dass das Formular (nach dem Show-Kommando) durch einen Mausklick wieder ausblendet.wird.

5.4 Übersicht über die Fenster im VBA-Editor

Die Abb. 6 zeigt alle Fenster, die bei den bis hierhin erfolgten Schritten im VBA-Editor entstanden sind:

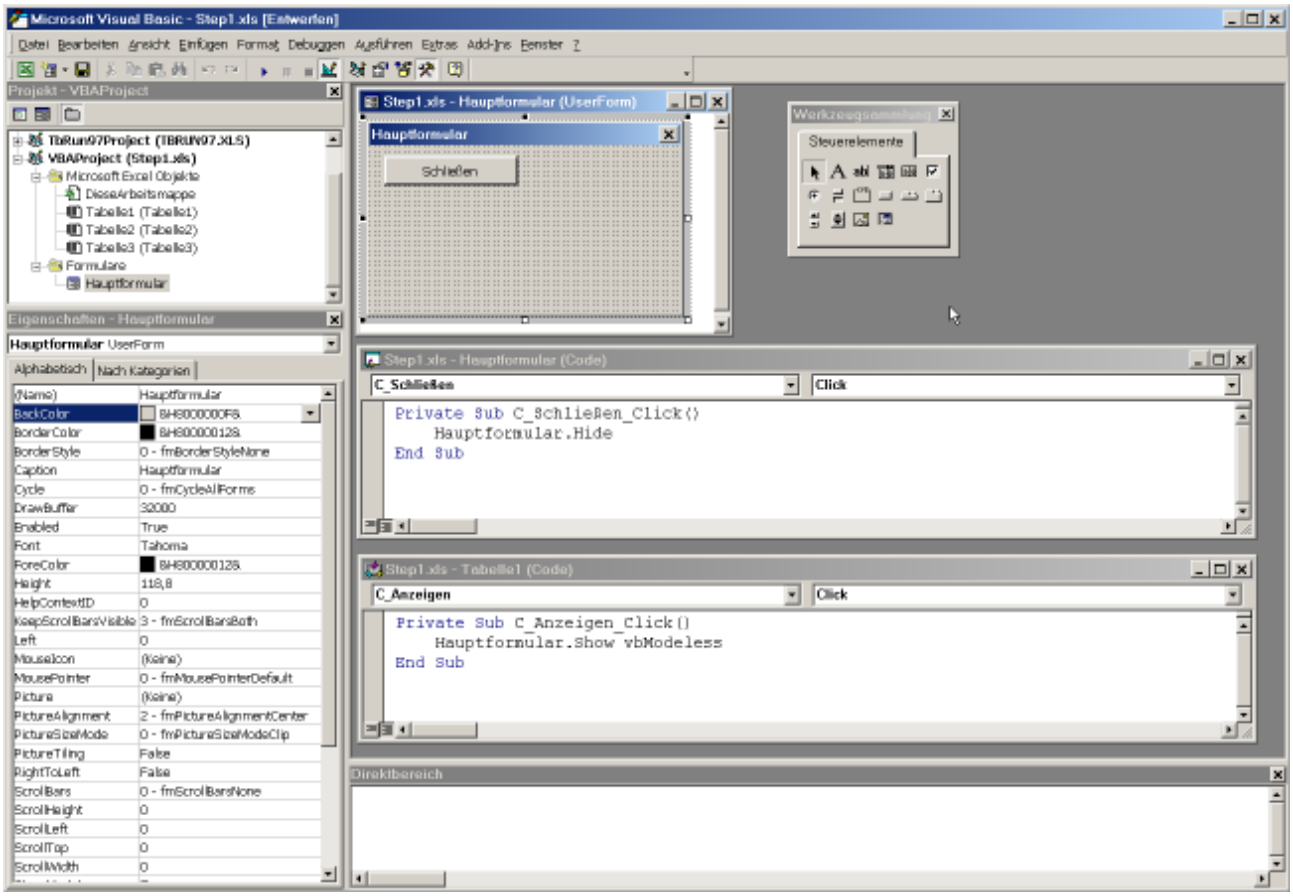

Abb. 6: Fenster im VBA-Editor für Anzeigen und Ausblenden des Formulars Hauptformular

Im linken oberen Drittel des Bildes sieht man die Einträge des Projekt-Explorerers, darunter das Eigenschaftsfenster. Da im Augenblick dieses Snapshots gerade die Objektanzeige des Formulars das aktivie Fenster ist, werden die Eigenschaften des Formulars und dazu die Werzeugleiste angezeigt - letztere wird automatisch ausgeblendet, wenn im Code-Fenster ein anderes Objekt angewählt wird, das kein Formular ist.

Im großen Code-Fenster werden neben der Werkzeugsammlung drei weitere Fenster angezeigt: Oben das Formular in der Objekt-Ansicht, darunter das Formular in der Code-Ansicht, wobei hier die Prozedur für die Schaltfläche C_Schließen wiedergegeben wird. Im unteren Fenster findet sich die Code-Ansicht des Tabellenblattes, wo für die Schaltfläche C_Anzeigen die zugehörige Prozedur aufgeführt wird.

Das Debug-Fenster („Direktbereich“) ist noch leer, das es bislang nicht verwendet wurde.

5.5 Verändern weiterer Eigenschaften der bislang vorhandenen Objekte

Um die Auswirkungen zu sehen, wenn man die Eigenschaften der Objekte verändert, bieten sich folgende Einträge - sofern für die jeweilige Objektklasse vorhanden - an (hier am Beispiel einer Schaltfläche):

BackColor	verändert die Hintergrundfarbe
BackStyle	verändert die Transparenz
ControlTipText	Text, der ausgegeben wird, wenn der Mauszeiger auf das Objekt bewegt wird
Font	Auswahl von Schriftart, Schriftgrad und Erscheinungsbild des Beschriftungstextes
ForeColor	Farbe des Beschriftungstextes
Height, Width	Höhe und Breite des Objektes in Rastereinheiten

6. Besonderheiten objektorientierter Programmierung

Im Gegensatz zur sequentiellen Programmierung, wie sie von einfachen Programmiersprachen wie Pascal, Basic oder C verwendet wird, steht die objektorientierte Programmierung wie z.B. in Delphi, Visual Basic oder C++. Pascal und Delphi, aber auch Basic und Visual Basic oder C und C++ gehören zwar jeweils zur gleichen Sprachklasse - was bedeutet, dass sie eng verwandte Befehle, Variablenbezeichner und Programmstrukturen kennen - unterscheiden sich aber ganz wesentlich in der Art der Ausführung der letztendlich damit geschriebenen Programme.

6.1 Quadratwurzelberechnung in sequentieller Programmierung

Als ein einfaches Beispiel soll die Eingabe einer Zahl, die Berechnung ihrer Quadratwurzel und die Ausgabe des Ergebnisses dienen.

6.1.1 Sequentielle Programmierung

Hier wird ein Schritt nach dem anderen ausgeführt. Das Programm stellt zunächst eine Eingabemöglichkeit zur Verfügung, in die der Benutzer eine Zahl eingibt und die Eingabe z.B. mit Enter bestätigt. Anschließend wird der Zahlenwert verwendet, um die Quadratwurzel zu berechnen. Das Ergebnis dieser Berechnung wird danach in Form einer Bildschirmausgabe dem Benutzer zur Verfügung gestellt. Anschließend ist das Programm beendet und muss für einen neuen Durchlauf erneut gestartet werden. Gemäß des Programmcodes müssen alle diese Schritte nacheinander erfolgen, insbesondere kann die Ausgabe nicht vor der Eingabe auf dem Bildschirm oder die Programmbeendingung nicht vor der Berechnung erfolgen.

6.1.2 Objektorientierte Programmierung

Für dieses Programm werden vier Objekte benötigt:

- ein Eingabefeld, das den Zahlenwert entgegennehmen kann
- ein Ausgabefeld, das das Ergebnis anzeigen kann
- eine Schaltfläche, die die Berechnung des Ergebnisses auslöst
- eine Schaltfläche, die das Programm beendet

Diese vier Objekte existieren gleichzeitig auf dem Bildschirm, was aber bedeutet, dass ein Klick auf die Schaltfläche die Berechnung des Ergebnisses auslösen kann, wobei es keine Rolle spielt, ob der Benutzer überhaupt schon einen Zahlenwert in das Eingabefeld eingetragen hat, ebenso kann das Programm jederzeit beendet werden, auch wenn es „noch gar nichts gemacht hat“.

6.2 Eigenschaften, Ereignisse und Methoden

Wie bereits in Kap. 1 kurz erläutert, verfügt jedes Objekt über **Eigenschaften**, die in der Objektklasse definiert sind und z.B. das Erscheinungsbild oder das grundsätzliche Verhalten des Objektes beeinflussen.

Der Benutzer als Bediener seines Computers löst bei jeder seiner Aktionen ein **Ereignis** aus, das kann z.B. eine Tastenbetätigung, ein Mausklick oder ein Bewegen der Maus sein. Aber auch andere Komponenten des Computers können Ereignisse auslösen, so z.B. das CD-ROM-Laufwerk oder der angeschlossene Drucker. Jedes Objekt verfügt deshalb über eine Liste von Ereignissen, auf die es in irgend einer Weise reagieren kann, wobei das übergeordnete Objekt entscheidet, ob das gerade aufgetretene Ereignis ignoriert, von ihm selbst bearbeitet oder an eines seiner untergeordneten Objekte weitergereicht wird.

Außerdem können Objekte ihrerseits bestimmte Ereignisse auslösen oder Funktionen ausüben. Die Liste aller dieser „aktiven Handlungen“ bezeichnet man als **Methoden**.

In der objektorientierten Programmierung ist es deshalb mit eine der wichtigsten Aufgaben des Programmierers, den von ihm verwendeten Objekten Anweisungen zu geben, auf welche Ereignisse sie reagieren sollen bzw. dürfen und welche Aktionen damit verbunden sein sollen. Die meisten Benutzeroberflächen zur objektorientierten Programmierung - so auch die von VBA - nehmen ihm dabei einen Großteil des dazu notwendigen Verwaltungsaufwandes ab (was sich allerdings manchmal auch als „Pferdefuß“ erweisen kann).

6.3 Quadratwurzelberechnung in objektorientierter Programmierung

Für dieses Beispiel werden die bereits unter 6.1.2 genannten Objekte benötigt. Diese befinden aich alle auf einem Formular (als übergeordnetes Objekt) einer Excel-Tabelle - doch auf diese Feinheiten soll nicht weiter eingegangen werden. Im Weiteren wird davon ausgegangen, dass dieses Formular mit dem Namen F_Quadratwurzel existiert und durch eine entsprechende Schaltfläche vom Tabellenblatt aus angezeigt werden kann.

6.3.1 Anlegen der benötigten Objekte

Wie in Kap. 3 beschrieben, werden nun auf dem Formular 2 Schaltflächen erzeugt:

(Name)	Caption
C_Berechnen	Berechnen
C_Schließen	Schließen

Für die Ein- und Ausgabe werden außerdem zwei Felder vom Typ TextBox benötigt, die ebenfalls auf dem Formular angelegt werden:

(Name)
T_Eingabe
T_Ausgabe

Das Formular könnte nun etwa wie in Abb. 7 aussehen.

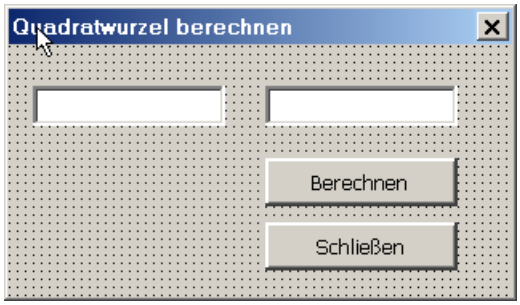

Abb. 7: Formular mit 4 Objekten

6.3.2 Prozeduren für die Schaltflächen erzeugen

Ein Mausklick auf die Schaltfläche C_Schließen soll das Formular ausblenden (vergl. Kap. 5.3), der zugehörige Code ist

```
Private Sub C_Schließen_Click()
    F_Quadratwurzel.Hide
End Sub
```

Ebenfalls nur eine Zeile wird benötigt, um die Quadratwurzel zu berechnen und auszugeben. Der zu verwendende Wert steht in der Textbox T_Eingabe, wobei deren Bezeichner wie ein Variablenbezeichner (z.B. x) verwendet werden kann.

Die Ausgabe soll in die Textbox T_Ausgabe erfolgen. Doch wie berechnet man die Wurzel einer Zahl?

Visual Basic „kennt“ einige mathematische Funktionen, die in der Sprachbibliothek gespeichert sind. Es sind deutlich weniger als z.B. bei einem guten Taschenrechner, doch kann man mit ihnen nahezu alle anderen benötigten Funktionen nachbilden. Um die Wurzel einer Zahl zu berechnen, kennt Visual Basic das Schlüsselwort Sqr. Wie bei einer mathematischen Funktion der Form y = f(x) wird dabei das Argument der Funktion in Klammern übergeben und der berechnete Wert mit einem Gleichheitszeichen der gewünschten Variablen zugewiesen. Folglich lautet der Code der entsprechenden Prozedur

```
Private Sub C_Berechnen_Click()
    T_Ausgabe = Sqr(T_Eingabe)
End Sub
```

6.3.3 Das Programm Quadratwurzelberechnung testen

Mit einem Klick auf den blauen Pfeil (s.a Kap. 5.2) oder mit der Funktionstaste <F5> kann nun getestet werden, ob alles funktioniert. Auf jeden Fall sollte das Formular angezeigt werden, wobei die beiden Textfelder noch leer sind, sonst stimmt evt. der Formularname nicht, der bei der „Anzeigen“-Schaltfläche des Tabellenblattes angegeben wurde.

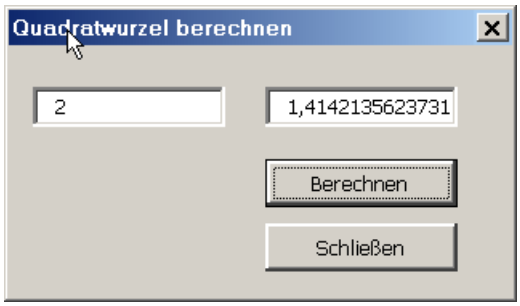

Abb. 8: Test der Quadratwurzellberechnung

Nun gibt man in das Eingabefeld eine Zahl ein, z.B. 2, und klickt auf „Berechnen“. Im Ausgabefeld sollte nun das Ergebnis erscheinen (vergl. Abb. 8). Man beachte, dass jederzeit - also auch vor Eingabe einer Zahl - die Schaltfläche „Schließen“ benutzt werden kann, um das Formular zu schließen.

6.4 Verwenden des Debug-Fensters („Direktbereich“)

Was aber geschieht, wenn man noch keinen Eintrag in das Eingabefeld gemacht (oder den Eintrag gelöscht) hat und man klickt auf „Berechnen“?

Es wird das VBA-Editor-Fenster eingeblendet, außerdem erscheint eine Meldung, die besagt, dass ein Fehler aufgetreten sei. Man hat nun die Wahl, mit „Beenden“ das Programm zu beenden oder in den „Debug“-Modus zu gehen. Klickt man auf diesen Button, verschwindet die Warnmeldung, stattdessen wird im Codefenster der Prozedur C_Berechnen_Click() die Kommandozeile gelb unterlegt. Bewegt man die Maus auf das Wort T_Eingabe, so erscheint T_Eingabe = "", was bedeutet, das dieses Feld leer ist. Die Funktion Sqr() benötigt aber einen Zahlenwert, andererseits kann man aber das Programm mit den Schaltflächen auf dem Formular nicht fortsetzen, solange der Debug-Modus aktiviert ist...

Um das Programm fortzusetzen, muss also der Eintrag in T_Eingabe geändert werden, dies geschieht nun im Debug-Fenster. Dort gibt man z.B. ein:

```
T_Eingabe = 2       <Enter>
```

(<Enter> heißt: Eingabe mit der Enter-Taste bestätigen.) Um sicher zu gehen, ob die Eingabe erfolgreich war, kann man sich in der nächsten Zeile des Debug-Fensters den neuen Wert von T_Eingabe auch anzeigen lassen:

```
? T_Eingabe         <Enter>
```

Ist die Ausgabe zufriedenstellend, drückt man nun die Taste <F5>, um zu testen, ob das Programm fortgesetzt werden kann. Ist alles in Ordnung, erscheint nun wieder das Formular mit dem eingegebenen Zahlenwert und dem Ergebnis im Ausgabefeld und man kann weiterarbeiten, ansonsten erfolgt erneut eine Fehlermeldung.

Ähnliches würde passieren, wenn man eine negative Zahl oder statt einer Zahl einen Text eingibt - in keinem der Fälle kann die Quadratwurzel berechnet werden.

6.5 Erste Schritte zur Eingabeabsicherung

Um im Betrieb eine solche Fehlermeldung wie oben beschrieben zu vermeiden, ist es Aufgabe des Programmierers dafür zu sorgen, dass diese Fehler vorher abgefangen werden können. Hier gibt es zwei Strategien, die sich gegenseitig ergänzen können:

- Aktionen können nur ausgeführt werden, wenn alle Vorbedingungen erfüllt sind.
- Fehlerbedingungen werden während der Laufzeit des Programmes erkannt und durch geeignete Meldungen oder Anweisungen behandelt.

6.5.1 Aktivieren / Deaktivieren einer Schaltfläche

Um z.B. zu verhindern, dass die Schaltfläche „Berechnen“ betätigt wird, während das Eingabefeld noch leer ist, kann folgende Überlegung weiter helfen:

„Wenn das Eingabefeld leer ist, dann deaktiviere die Schaltfläche „Berechnen“, sonst aktiviere sie“

Da VBA (wie fast alle Programmiersprachen) auf die englische Sprache zurückgreift, um den Sprachwortschatz darzustellen, lautet diese Bedingung in VBA:

```
If T_Eingabe = "" Then C_Berechnen.Enabled = False Else C_Berechnen.Enabled = True
```

Die Fehlerbedingung hinter dem Schlüsselwort If orientiert sich an der Fehlermeldung (s.o.), dabei zeigt "" eine leere Zeichenkette an (zwischen den Anführungszeichen darf kein Leerzeichen stehen!).

Nach dem Schlüsselwort Then folgt die in diesem Fall auszuführende Anweisung - hier wird die Eigenschaft Enabled des Objektes C_Berechnen so eingestellt, dass es auf nicht (False) auf einen Mauslklick reagiert.

Falls diese Bedingung nicht erfüllt ist, folgt nach dem Schlüsselwort Else die entsprechende Anweisung, hier wird die Schaltfläche aktiviert (True).

Die beiden Schlüsselworte True und False sind in VBA als Datentyp Boolean fest definiert. Da die Eigenschaft Enabled ebenfalls vom Typ Boolean ist, kann ihr nur einer dieser beiden Werte - also z.B. keine Zahl - zugewiesen werden.

6.5.2 Überprüfen der Zulässigkeit der Eingabe

Die Quadratwurzel kann nur aus einer Zahl gezogen werden, die größer oder gleich 0 ist. Gibt der Anwender z.B. die Zahl -2 ein, ist das Eingabefeld nicht mehr leer und die Sicherheitsabfrage, wie sie in 6.5.1 beschrieben wurde, greift nicht mehr - die Schaltfläche würde aktiviert werden.

Um zu verhindern, dass die Eingabe einer negativen Zahl zu einem Programmfehler führt, muss also überprüft werden, ob die eingegebene Zahl kleiner als 0 ist. Auch dies lässt sich als Bedingung darstellen:

"Wenn die Zahl im Eingabefeld kleiner als 0 ist, dann ..."

In der Sprache von VBA ergibt sich daraus

```
If T_Eingabe < 0 Then (...)
```

wobei (...) durch die entsprechende Anweisung ergänzt werden muss:

```
If T_Eingabe < 0 Then C_Berechnen.Enabled = False Else C_Berechnen.Enabled = True
```

6.5.3 Positionierung der Sicherheitsabfragen

Um festzulegen, an welcher Stelle diese Sicherheitsabfragen eingebaut werden können, müssen die Eigenarten objektorientierter Programmierung berücksichtigt werden.

Platziert man sie in der Prozedure C_Berechnen_Click(), würde man die Schaltfläche deaktivieren, während bereits der zugehörige Code ausgeführt wird - damit würde aber direkt anschließend die Berechnung erfolgen und zu einer Fehlermeldung führen, die man ja gerade vermeiden wollte.

Wie in Kap. 6.2 beschrieben verfügt aber jedes Objekt über Methoden, mit denen es auf bestimmte Ereignisse reagieren kann. So reagiert z.B. das Textfeld T_Eingabe auf das Ereignis Change durch Aufrufen der Prozedur T_Eingabe_Change(), welches auftritt, wenn das Feld z.B. durch eine Eingabe verändert wird.

Die beiden in 6.5.1 und 6.5.2 beschriebenen Überprüfungen des Zustandes von T_Eingabe sollten immer dann vorgenommen werden, wenn eine Änderung des Feldinhaltes aufgetreten ist. Damit könnte sich für die zugehörige Prozedur T_Eingabe_Change() folgende Form ergeben:

```
Private Sub T_Eingabe_Change()
    If T_Eingabe = "" Then C_Berechnen.Enabled = False Else C_Berechnen.Enabled = True
    If T_Eingabe < 0 Then C_Berechnen.Enabled = False Else C_Berechnen.Enabled = True
End Sub
```

Eine genauere Betrachtung zeigt jedoch, dass diese Form der beiden aufeinanderfolgenden If ... Then ... Else ... Abfragen ungeeignet ist, da die letzte Abfrage das Ergebnis der ersten überschreibt. Eine genauere Analyse der möglichen Bedingungen führt aber zu folgender Aussage:

"Wenn das Eingabefeld leer oder dessen Inhalt eine Zahl kleiner als 0 ist, dann deaktiviere die Schaltfläche „Berechnen", sonst aktiviere sie."

Auch diese verbale Formulierung der nun aus zwei Teilbedingungen zusammengesetzten Bedingung lässt sich unmittelbar in VBA übersetzen:

```
If (T_Eingabe = "") Or (T_Eingabe < 0) _
    Then C_Berechnen.Enabled = False _
    Else C_Berechnen.Enabled = True
```

(Man beachte den Unterstrich _ am Ende der ersten beiden Zeilen: normalerweise müsste diese Anweisung in einer Zeile geschrieben werden, reicht der Platz jedoch nicht aus, kann man damit VBA mitteilen, dass die Befehlszeile in der nächsten Bildschirmzeile noch fortgesetzt wird und keine eigene Anweisung darstellt!)

Somit ergibt sich nun für die Prozedur T_Eingabe_Change() folgender Aufbau:

```
Private Sub T_Eingabe_Change()
    If (T_Eingabe = "") Or (T_Eingabe < 0) _
        Then C_Berechnen.Enabled = False _
        Else C_Berechnen.Enabled = True
End Sub
```

6.5.4 Testen der Sicherheitsabfragen

1. Feststellung:

Nach dem Öffnen des Formulars ist das Eingabefeld leer, trotzdem ist die Schaltfläche „Berechnen" noch aktiviert.

Abhilfe: Beim Programmstart muss die Schaltfläche deaktiviert sein, dazu ändert man im Bearbeiten-Modus die zugehörige Eigenschaft Enabled von C_Berechnen auf False um.

2. Feststellung:

Bei Eingabe eines Minuszeichens wird die Schaltfläche aktiviert, obwohl noch keine Ziffer eingegeben wurde.

Abhilfe: Ein Minuszeichen allein darf noch nicht zu einer Aktivierung der Schaltfläche führen.

Dazu ist die Bedingung in der Sicherheitsabfrage um eine weitere Teilbedingung zu ergänzen:

```
If (T_Eingabe = "-") Or ...
```

3. Feststellung:

Bei Eingabe einer oder mehrerer Ziffern wird die Schaltfläche aktiviert und bleibt auch dann aktiviert, wenn statt einer Zahl ein Buchstabe eingegeben wird, obwohl es sich nun eindeutig um keine Zahl mehr handelt.

Abhilfe: Die Überprüfung des Feldinhaltes muss auf für Zahlen unzulässige Zeichen erweitert werden.

Diese Überprüfung bedarf weitergehender Überlegungen, da hier eine nahezu unendliche Zahl von Möglichkeiten an Eingabefehlern besteht. Doch auch hier hilft die Sprachbibliothek von VBA etwas weiter: Zum Bestimmen des Zahlenwertes einer Eingabe kennt VBA das Schlüsselwort Val(), wobei als Argument eine Zeichenfolge, also z.B. der Inhalt eines Textfeldes übergeben wird. Diese Funktion arbeitet die Zeichenfolge von links nach rechts ab und bricht die Bearbeitung ab, wenn das erste Zeichen auftritt, das nicht Bestandteil einer Zahl sein kann. Solange man nur mit ganzen Zahlen arbeitet, stimmt die von Val() übergebene Zahl mit der Zeichenfolge des übergebenen Textfeldes überein, allerdings scheitert das Verfahren bei Dezimalzahlen oder Zahlen in exponentieller Darstellung, wie man an folgenden Beispielen erkennen kann:

T_Eingabe	Val(T_Eingabe)	Übereinstimmung
123	123	Ja
12x	12	Nein (das x wird nicht mehr ausgewertet)
2,3	2	Nein (das Komma wird als Trennzeichen interpretiert)
2.3	2,3	Nein (der Dezimalpunkt wird in ein Komma umgewandelt)
2E3	2000	Nein (der Exponent wird ausgewertet)
2E35	2E+35	Nein (es wird ein Pluszeichen hinzugefügt)
2E+35	2E+35	Ja
2.3E+35	2,3E+35	Nein (der Dezimalpunkt wird in ein Komma umgewandelt)

Hinzu kommt, dass die Funktion Sqr() Dezimalzahlen erwartet, die ein Komma aufweisen, die Funktion Val() aber hierfür einen Punkt verwendet. Mit Val() kann also weder eine Übereinstimmung für Dezimalzahlen überprüft noch kann Val() verwendet werden, um die Zahleneingabe für das Argument von Sqr() aufzubereiten.

Zumindest für die Überprüfung ganzer Zahlen wäre aber nach obigen Überlegungen folgende Teilbedingung als Ergänzung für die Sicherheitsabfrage geeignet (<> ist der Vergleichsoperator für "ungleich"):

```
If (T_Eingabe <> Val(T_Eingabe)) Or ...
```

(Man beachte die Klammerebenen: das äußere Klammerpaar umschließt den Vergleich, dessen Ergebnis True oder False ist, während das innere Klammerpaar das Argument der Funktion Val() einschließt. Obwohl es eine Hierarchie in der Abarbeitung logischer Operatoren gibt, erleichtern diese Klammern die Lesbarkeit und helfen Fehler zu vermeiden.)

Da ein einzelnes Minuszeichen durch diese Abfrage ebenfalls erfasst wird, kann nun die Teilbedingung, mit der der Fehler der 2. Feststellung behoben wurde, wieder entfernt werden.