Objekte, Eigenschaften und Methoden

Im Vorwort und in der Einleitung ist öfter schon mal das Wort Objekt gefallen. Mit Visual Basic führte Microsoft das Prinzip der objektorientierten Programmierung (OOP) ein, und dieses Prinzip verwenden mittlerweile die meisten Programmiersprachen. Nehmen Sie sich die Zeit und lesen Sie in diesem Kapitel das Wichtigste zum Thema Objekte, Eigenschaften und Methoden. Das Wissen brauchen Sie, um die Aufzeichnungen des Makrorekorders zu verstehen und komplexere Makros zu schreiben als *Hallo Welt* und *Alter berechnen*.

3.1 Alles ist Objekt

Mit OOP wird alles, was für die Programmierung relevant ist, als Objekt behandelt. Das beginnt bei Excel selbst (ja, Excel ist auch ein Objekt) und geht weiter zu Mappen, Tabellenblättern, Zellen und Diagrammen, um mal die wichtigsten Objekte zu nennen. Jedes Objekt kann weitere Objekte enthalten, und das nennt man dann Objekthierarchie. Die Mutter aller Objekte heißt Application (richtig: Excel himself). So sieht die Hierarchie mit einigen Unterobjekten aus:

Application
- AddIn
- Window
- Workbook
- WorksheetFunction

Das Objekt Workbook steht für die Arbeitsmappe, und die hat weitere Objekte, zum Beispiel:

Workbook
- Worksheet
- Chart
- Name

Und so geht es weiter, das Worksheet-Objekt (Tabellenblatt) hat u. a. diese Objekte:

Worksheet
- Comment
- Hyperlink
- Name
- PivotTable

Um ein Objekt in einem Objekt zur Programmierung zu benutzen, müssten Sie nichts anderes tun, als sich in der Objekthierarchie bis zu dem Objekt durchzugraben. Beispiele:

Objekt	Objekt (in VBA)
Eine Arbeitsmappe	`Application.Workbook`
Ein Tabellenblatt	`Application.Workbook.Worksheet`
Ein Bereich in einem Arbeitsblatt	`Application.Workbook.Worksheet.Range`
Ein Diagrammobjekt	`Application.Workbook.Worksheet.Chartobject`

Objekte behalten das, was sie haben, über die gesamte Lebensdauer, die sich meist auf die Laufzeit eines Makros beschränkt. Deshalb werden Objektvariablen am Ende eines Makros immer mit *Nothing* belegt, um den Speicher zu leeren, den sie belegt hatten:

```
Set <Objekvariable> = Nothing
```

3.2 Collections

Was tun, wenn mehrere Arbeitsmappen unter Excel offen sind? Wenn eine Mappe mehrere Tabellenblätter hat (was ja meist der Fall ist) oder wenn ein Tabellenblatt mehrere Diagrammobjekte enthält? Dafür bietet VBA das geniale Konzept der Collections. Eine Collection (Sammlung) ist eine Gruppe von Objekten desselben Objekttyps. Die wichtigsten und häufigsten Collections sind diese:

- Workbooks — Alle offenen Arbeitsmappen
- Worksheets — Alle Tabellenblätter einer Arbeitsmappe
- Charts — Alle Diagrammblätter in einer Arbeitsmappe
- Chartobjects — Alle Diagrammobjekte auf einem Tabellenblatt

Um im VBA-Makro mit Collections zu programmieren, gibt es zwei Möglichkeiten:

- Sie benutzen die Collection selbst und verwenden eine Eigenschaft oder eine Methode. Beispiel:

  ```
  Workbooks.count
  ```

  Ergebnis: Die Anzahl aktiver Arbeitsmappen

- Sie indizieren eine Collection und verwenden ein Element daraus. Dazu geben Sie den Namen des Elements oder seine Nummer an:

```
Workbooks("Einkauf.xlsx") oder
Workbooks(1)
```

In der Objekthierarchie sieht das Ganze dann so aus:

Objekt	Objekt (in VBA)
Die Arbeitsmappe *Einkauf.xlsx*	`Application.Workbooks("Einkauf.xlsx")`
Das Tabellenblatt "Filiale 1" in der Arbeitsmappe "Einkauf.xlsx"	`Application.Workbooks("Einkauf.xlsx").Worksheets("Filiale 1")`
Der Bereich "$A$1:$C$20" im Tabellenblatt "Filiale 1" in der Arbeitsmappe "Einkauf.xlsx"	`Application.Workbooks("Einkauf.xlsx").Worksheets("Filiale 1").Range("$A$1:$C$20")`
Das Diagrammobjekt "Diagramm 1" im Tabellenblatt "Filiale 1" in der Arbeitsmappe "Einkauf.xlsx"	`Application.Workbooks("Einkauf.xlsx").Worksheets("Filiale 1").Chartobjects("Diagramm 1")`
Das erste Diagrammobjekt im ersten Tabellenblatt in der ersten Arbeitsmappe	`Application.Workbooks(1).Worksheets(1).Chartobjects(1)`

Der wichtigste Unterschied bei der Indizierung von Collections: Verwenden Sie einen Namen, geben Sie diesen immer in Anführungszeichen ein. Verwenden Sie Position oder Nummer, tragen Sie diese ohne Anführungszeichen zwischen die Klammern ein. Darüber hinaus gibt es noch weitere Möglichkeiten, ein Objekt in einer Collection zu indizieren, zum Beispiel:

Ausdruck	Beschreibung
`Worksheets(shName)`	Erlaubt, wenn *shName* ein Bereichsname mit Bezug auf einen Tabellenblattnamen ist.
`Worksheets(range("$A$1").Value)`	Erlaubt, wenn in Zelle A1 der Name eines Tabellenblatts steht.

`Dim strTab`
`strTab = "Filiale 1"`
`Worksheets(strTab).Select`
	Hier wird eine Variable angelegt und mit dem Namen eines Tabellenblatts versehen. Die Variable kann dann zur Indizierung der Collection *Worksheets* verwendet werden.

3.2.1 Objektreferenzen vereinfachen

Keine Angst: In der Praxis werden Sie selten lange Monsterhierarchien schreiben müssen. Befinden Sie sich zum Beispiel beim Makrostart sicher in der aktiven Mappe, reicht es, wenn Sie den Namen des Tabellenblatts angeben. Und wenn Sie per Makro in ein Tabellenblatt wechseln oder ein neues Blatt anlegen, müssen

Sie dieses nicht mehr angeben, um einen Bereich oder ein Diagrammobjekt anzuprogrammieren. Das Objekt *Application* können Sie fast immer weglassen, außer es besteht die Gefahr, dass eine andere Application (Word, PowerPoint) ins Spiel kommt und das Makro von dieser aus ausgeführt wird.

Schreiben Sie doch einmal die Zahl *12345* in die Zelle A1 eines Tabellenblatts und lesen Sie die Zelle per Makro aus. Die lange Version sieht so aus:

```
Sub Zelle_A1_Auslesen
  MsgBox Application.Workbooks("Einkauf.xlsx").Worksheets("Filiale 1").Range("$A$1")
End Sub
```

Lassen Sie Ihr Makro vorher durch die Objekthierarchie wandern, wird das Ganze einfacher und Sie können, um weitere Zellen zu beschriften, das letzte Objekt (Range) verwenden:

```
Sub Zelle_A1_Auslesen_2
  Application.Workbooks("Einkauf.xlsx").Activate
  Worksheets("Filiale 1").Select
  MsgBox Range("$A$1").Value
End Sub
```

3.2.2 With und End With

Sehr hilfreich in diesem Zusammenhang ist eine Kontrollstruktur, die auch der Makrorekorder häufig einsetzt, um sich beim Aufzeichnen von Makros lange und überflüssige »Objektbäume« zu sparen. Mit *With* wird das Objekt eingeführt, auf das sich alle weiteren Anweisungen beziehen. *End With* beendet das Ganze wieder:

```
With Objekt
         .Objekt.Eigenschaft.Methode
         .Objekt.Eigenschaft.Methode
         .Objekt.Eigenschaft.Methode
End With
```

So sieht das in unserem Beispiel aus:

```
With Application.Workbooks("Einkauf.xlsx").Worksheets("Filiale 1")
         MsgBox .Range("$A$1").Value
End With
```

Haben Sie den wichtigen Unterschied bemerkt? Wenn das Objekt bzw. die Objekthierarchie mit *With* »adressiert« wird, muss das Makro nicht die ganze Hierarchie durchwandern, um zum begehrten Zielobjekt zu gelangen. Die Zelle A1 kann im obigen Beispiel ausgelesen werden, ohne dass das Makro die Mappe öffnet und in das Tabellenblatt wechselt.

3.2.3 Objekteigenschaften und Methoden

Wie Sie in den Beispielen zu den Objekten schon gesehen haben, reicht es nicht, die Objekthierarchie einfach anzugeben, egal, wie groß sie ist und wie viele Objekte sie enthält. In VBA wird mit den Objekten immer etwas Praktisches passieren. Arbeitsmappen werden geöffnet, aktiviert oder geschlossen (oder auch nur gezählt), Tabellenblätter aktiviert man per VBA, benennt sie um oder löscht sie und Bereiche (Zellen) bekommen per Makro Inhalte, Zahlenformate, Farben und Rahmen. Unterscheiden wir zwischen Eigenschaften und Methoden:

- Eigenschaften eines Objekts werden ausgelesen, festgelegt oder geändert.
- Methoden führen eine Aktion mit einem Objekt durch.

Jedes Objekt hat Eigenschaften und Methoden, und nicht selten haben unterschiedliche Objekte die gleichen Eigenschaften oder Methoden. Die Eigenschaft *Name* dürfte die häufigste im Objektkatalog sein, so ziemlich jedes Objekt hat nämlich einen Namen:

```
Workbook(1).Name
Worksheets(1).Name
ChartObjects(5).Name
usw.
```

Die häufigste Methode ist *Select*, sie wird zum Beispiel verwendet, um Tabellenblätter anzusteuern oder Zellbereiche zu markieren:

```
Worksheets("Filiale 1").Select
Range("$A$1:$C$20").Select
ChartObjects(1).Axis(1).Select
```

Eine Liste mit allen Objekten und deren Eigenschaften und Methoden anzufertigen, ist unmöglich, es gibt zu viele Objekte, und die haben oft die gleichen Eigenschaften und Methoden.

3.2.4 Ereignisse

Als VBA-Programmierer arbeiten Sie nicht nur mit Objekten (inklusive Eigenschaften, Methoden), sondern auch mit Ereignissen. Ein Ereignis ist zum Beispiel das Öffnen einer Mappe, der Wechsel auf ein anderes Tabellenblatt oder das Markieren eines Zellbereichs. Selbst der Mausklick auf eine Zelle löst bereits ein Ereignis aus. Ereignismakros werden häufig eingesetzt, um Prozesse zu automatisieren und abzusichern, aber auch, um Daten bereitzustellen oder Oberflächen zu präparieren.

3.2.5 Das Objektmodell »Autohaus«

Haben Sie das Objektmodell verstanden? Hier noch einmal eine Zusammenfassung und ein praktischer Vergleich, sagen wir mit der *Application* »Autohaus«:

	Im Objektmodell	Im Beispiel
Das übergeordnete Objekt	Excel	Ein Autohaus
Objekte, Collections	Arbeitsmappen (Workbooks) Tabellenblätter (Worksheets) Bereiche (Ranges) Grafische Objekte (Shapes)	Eine Modellreihe Einzelne Fahrzeuge der Modellreihen Varianten der Fahrzeugmodelle

Eigenschaften	Anzahl offener Arbeitsmappen:
	`Workbooks.count`
	Anzahl Tabellenblätter:
	`Workbooks(1).Sheets.count`
	Name eines Tabellenblatts:
	`Worksheets(1).Name`
		Anzahl der Fahrzeuge im Autohaus
		Namen der Modellreihen
		Eigenschaften des Objekts Fahrzeug:
		Modellbezeichnung, Motorisierung, Farbe, Ausstattung, Polsterung

Methoden	Neue Arbeitsmappe anlegen:
	`Workbooks.Add`
	Neues Tabellenblatt anlegen:
	`Sheets.Add`
	Erstes Tabellenblatt löschen:
	`Sheets(1).Delete`
	Tabellenblatt 5 an den Anfang verschieben:
	`Worksheets(5).Move Before:=1`
		Neue Baureihe einführen, neue Fahrzeuge ausstellen, Fahrzeuge verkaufen
		Methoden des Objekts Fahrzeug:
		Fahrzeug ausstatten, Fahrzeug umparken, Fahrzeug starten, fahren

Im Autohaus würde der Besitzer sagen: Ich habe für mein Autohaus (Application) die Baureihe G eingeführt (Objekt), zwei Fahrzeuge, einen GLK und einen GLA gekauft (Objekte) und beide im Autohaus geparkt (Methode). Der GLK ist tenoritgrau (Eigenschaft), er hat ein Automatikgetriebe (Eigenschaft). Der GLA wurde verkauft (Methode), er hatte ein Schiebedach (Eigenschaft).

In Excel sieht das so aus: Ich habe in Excel (Application) eine Arbeitsmappe (Objekt) angelegt, sie enthält drei Tabellenblätter (Objekte), das erste hat die Bezeichnung »Einkauf« (Eigenschaft), das zweite »Verkauf« (Eigenschaft). Das zweite Blatt habe ich vor das erste geschoben (Methode), das dritte gelöscht (Methode).

3.3 Der Objektkatalog

Obwohl es wie schon erwähnt unmöglich ist, alle Objekte mit ihren Eigenschaften und Methoden zu listen, bietet der Visual Basic Editor einen Objektkatalog an.

Wählen Sie *Ansicht/Objektkatalog* oder drücken Sie F2. Der Objektkatalog wird in einem neuen Fenster aktiviert, um ihn wieder zu schließen, klicken Sie auf das Schließen-Symbol rechts oben.

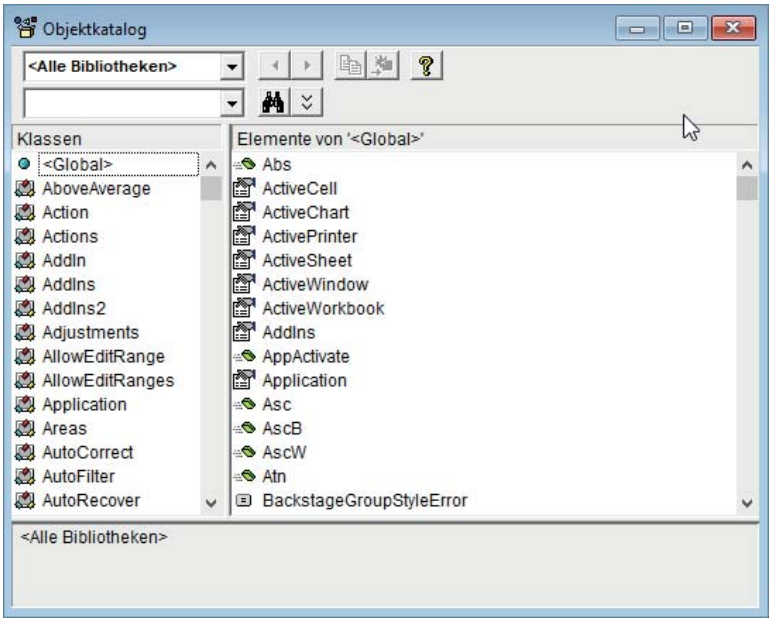


Abbildung 3.1: Der Objektkatalog.

Welche Objekte der Objektkatalog anzeigt, hängt von der Einstellung in der ersten Auswahlliste ab. Steht da *Alle Bibliotheken*, erhalten Sie eine Übersicht über alle Objekte in allen Objektbibliotheken. Das ist zunächst nicht sinnvoll, weil der Katalog a) viel zu groß ist und b) viele Einträge enthält, die Sie niemals brauchen werden. Schalten Sie deshalb auf den Eintrag *Excel* um. Jetzt sehen Sie eine vollständige Liste aller Excel-Objekte mit ihren Eigenschaften und Methoden.

Klicken Sie links auf ein Objekt, sehen Sie rechts alle Eigenschaften, Methoden und Ereignisse, die diesem Objekt zugewiesen sind. Am Symbol erkennen Sie, worum es sich handelt.

Symbol	Bedeutung
	Eigenschaft
	Methode
	Ereignis

Hier zum Beispiel das Objekt *Application*. Die Eigenschaften *ActiveWorkbook* und *ActivePrinter* erklären sich von selbst, mit der Methode *Calculate* könnten Sie eine Neuberechnung starten, und damit würden Sie das Ereignis *SheetCalculate* auslösen.

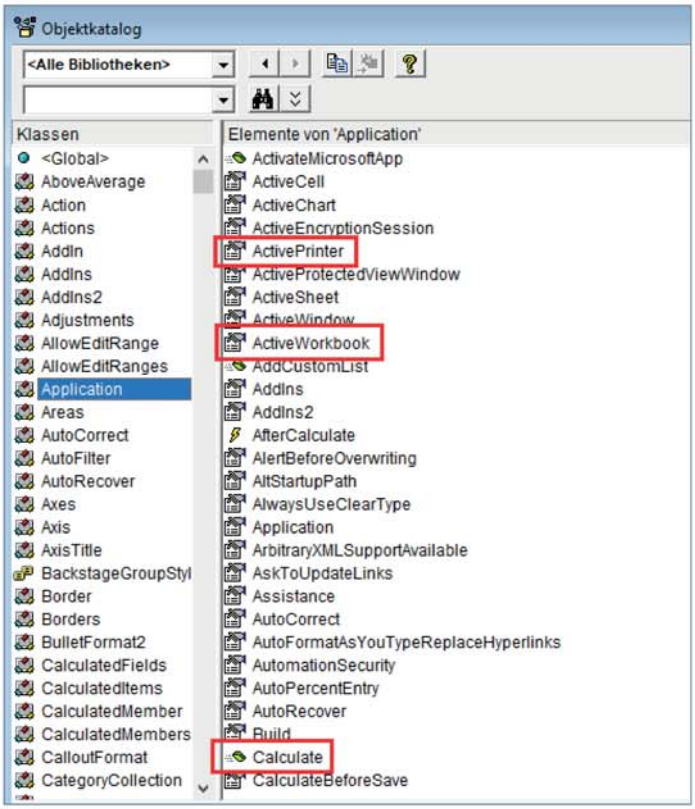


Abbildung 3.2: Objekte, Eigenschaften und Methoden.

Im Objektkatalog werden die Objekte als Klassen bezeichnet, weil sie in den Bibliotheken als solche programmiert sind. Eine Klasse enthält nicht nur den Objektnamen, sondern auch alle Eigenschaften und Methoden. Neben den Standard-

Elementen finden Sie in einigen Bibliotheken auch Konstanten, Module, Klassen, benutzerdefinierte Typen und Aufzählungen.

3.4 Objektsuche und Objekthilfe

Geben Sie in das zweite Listenfeld einen Suchbegriff ein, um nach Einträgen im Katalog zu suchen. Geben Sie beispielsweise *Comment* ein und bestätigen Sie mit ↵, zeigt der Katalog alle Einträge, in denen dieser Text vorkommt.

Mit dem Hilfe-Symbol erhalten Sie eine schnelle Beschreibung zum jeweils markierten Element im Objektkatalog. Dazu aktiviert der VBA-Editor Ihren Browser und schaltet auf die VBA-Referenz im Office-Dev-Center (MSDN) um. In der Regel ist neben der Beschreibung auch ein kleines Beispielmakro enthalten.

Abbildung 3.3: Das Hilfesymbol führt zu MSDN.

3.5 Objektbibliotheken und Verweise

Wenn Sie einen Blick in die Liste der Objektkataloge werfen (Auswahlliste links oben im Objektkatalog), sehen Sie, dass neben den Hauptbibliotheken Excel und VBA noch weitere Bibliotheken verfügbar sind. Aber das sind längst noch nicht alle. Wählen Sie *Extras/Verweise*, erhalten Sie die Auswahl aller verfügbaren Objektbibliotheken.

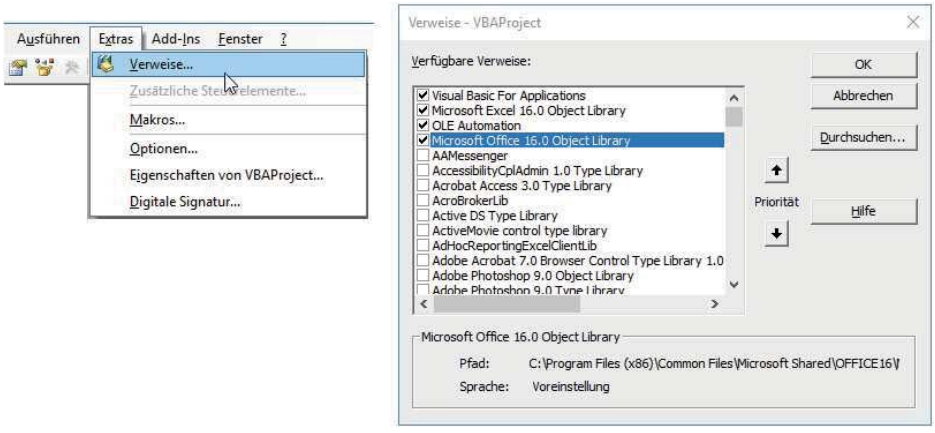


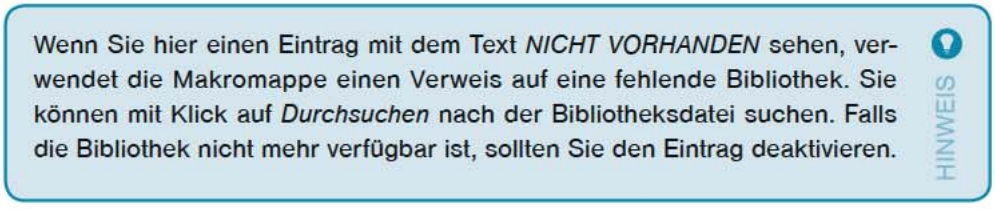
Abbildung 3.4: Die Objektbibliotheken unter Extras/Verweise.

Klicken Sie eine Bibliothek an, zeigt das Verweisfenster im unteren Bereich, aus welcher Datei diese stammt. Die Excel-Bibliothek ist tatsächlich in der Programmdatei EXCEL.EXE selbst hinterlegt, die VBA-Objekte stammen aus einer DLL (Dynamic Link Library), andere Bibliotheken aus Dateien mit der Endung TLB. Diese vier Bibliotheken sind standardmäßig aktiv:

Bibliothek	Beschreibung	Datei
OLE Automation	Basisfunktionen für ActiveX	StdOle2.tlb
Visual Basic for Applications	Die VBA-Objektbibliothek	VBE7.DLL
Microsoft Excel 16.0 Object Library	Die Excel-Objektbibliothek	EXCEL.EXE
Microsoft Office 16.0 Object Library	Die Bibliothek mit gemeinsamen Objekten aller Office-Komponenten	Mso.dll

Die Bibliothek *Microsoft Forms 2.0 Object Library* wird automatisch aktiviert, wenn Sie in der Mappe mit Dialogen (UserForms) arbeiten.

Um in VBA-Makros mit Objekten aus anderen Programmen oder Datenumgebungen zu arbeiten, wird also einfach die passende Bibliothek eingeschaltet. Programmieren Sie beispielsweise mit einem Excel-VBA-Makro an einem Word-Dokument, verweisen Sie auf die Bibliothek *Microsoft Word 16.0 Object Library*, und mit der *Microsoft PowerPoint 16.0 Object Library* stehen Ihnen alle Objekte aus PowerPoint zur Verfügung.



Schreiben Sie ein Makro, das alle aktiven Verweise auflistet. Sie brauchen zwei Variablen, eine Objektvariable für das Projekt und eine Schleifenvariable. Das Makro legt zunächst ein neues Tabellenblatt an, beschriftet die ersten drei Spalten und startet dann eine Schleife über alle Verweise. Der Schleifenzähler sorgt dafür, dass jeder Verweis eine eigene Zeile bekommt:

```
Cells(schleifenzähler + 1, spaltennummer)
```

Und aus dem Containerobjekt *References* werden die Eigenschaften *Name*, *Description* und *Fullpath* übernommen. Zum Schluss bekommen die drei Spalten noch eine optimale Spaltenbreite.