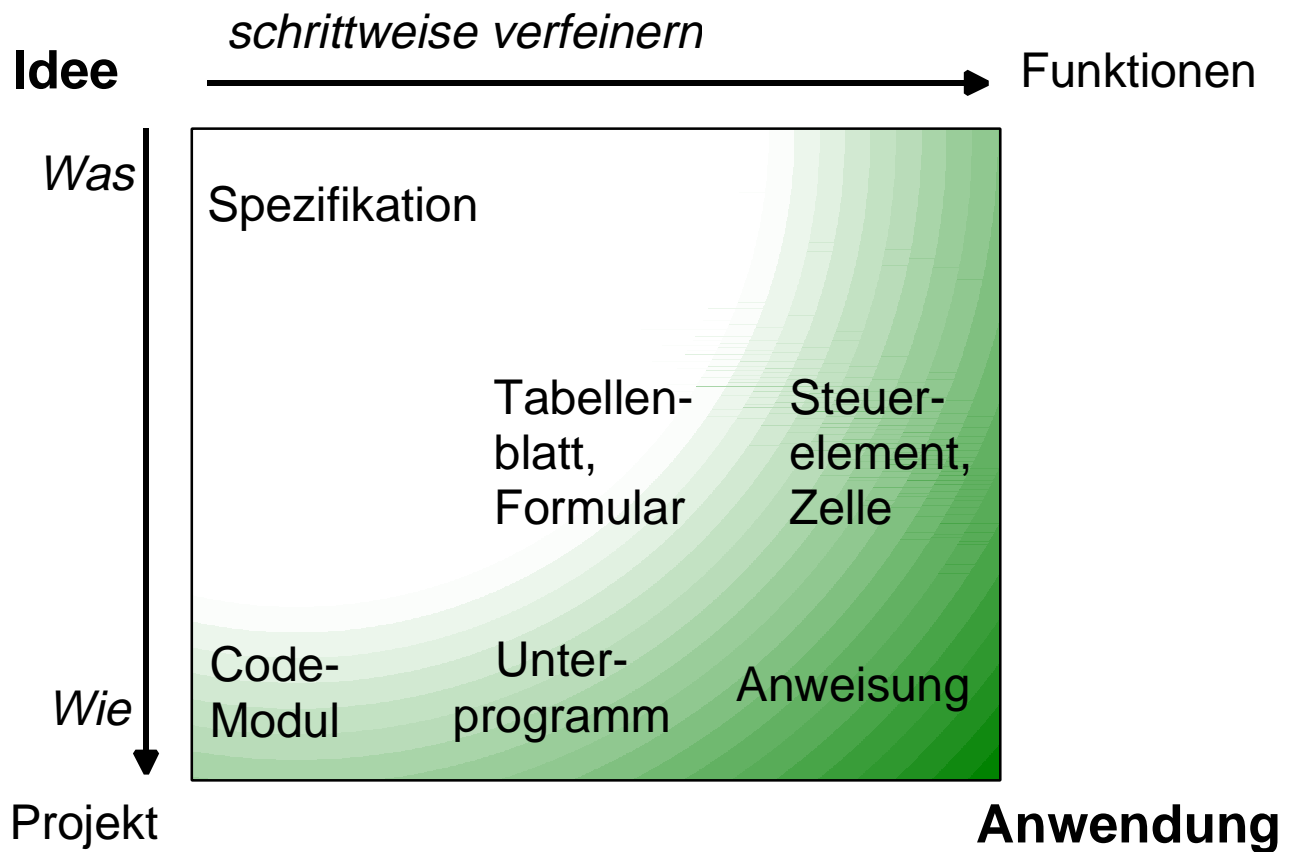


Entwicklungswerkzeuge



Werkzeuge eines Entwicklers

- a) Tabellenblatt und Formulare
- b) Programmeditor
- c) Projektextplorer
- d) Textverarbeitung
- e) Debugger
- f) Papier und Bleistift

Excel und VBA als Entwicklungsumgebung

Entwicklung eines Programms in **vier** Schritten:

- 1) grafische **Benutzerschnittstelle** gestalten
 - Formular-Symbolleiste (Steuerelemente, engl. controls)
 - Tabellenblätter, Zellen und Zellbereiche

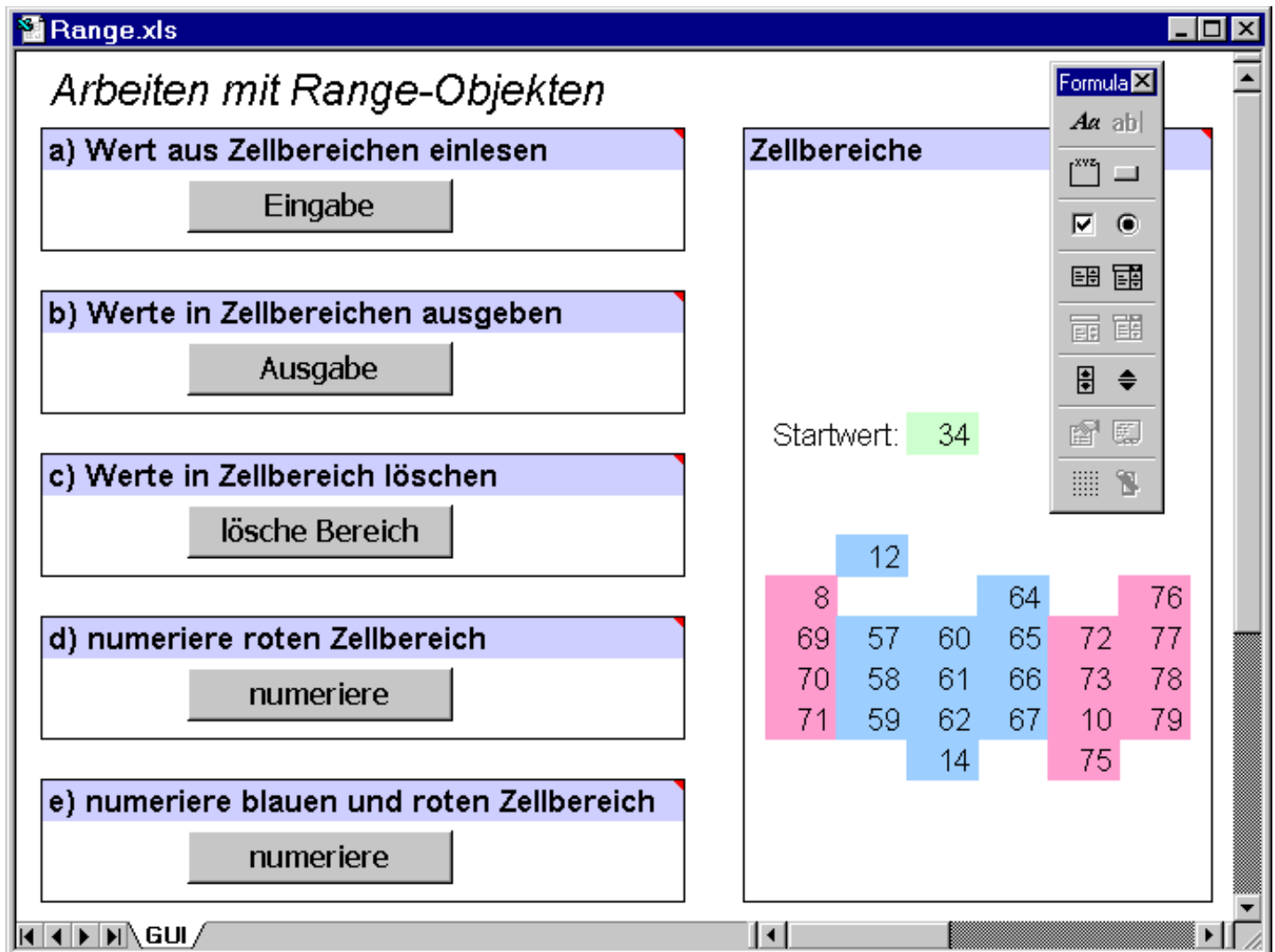
- 2) **Eigenschaften** der Schnittstellenelemente festlegen
 - Maus
 - Eigenschaftsfenster

- 3) **Verhalten** programmieren
 - Projekt-Explorer
 - Editor
 - Bibliotheken
 - Dokumentationen

- 4) Programm **testen** und optimieren
 - Debugger

Programmbeispiel Range.xls

1) Benutzerschnittstelle gestalten



Eingabe... **Z**ellbereiche (benannt)

Verarbeitung... **S**teuerelemente (v.a. Schaltflächen)

Ausgabe... **Z**ellbereiche (benannt)

a) Eingabe von Werten

Startwert: 34

		2				
8			64		76	
69	57	60	65	72	77	
70	58	61	66	73	78	
71	59	62	67	10	79	
		14		75		

Eingabe

```
Nummer = Application _  
    .Workbooks("Range.xls") _  
    .Worksheets("GUI") _  
    .Range("roterBereich") _  
    .Cells(2,3) _  
    .Value
```

```
Nummer = Tabelle1 _  
    .Range("roterBereich") _  
    .Cells(3,5) _  
    .Value
```

```
Nummer = Range("roterBereich").Cells(1,3).Value
```

```
Nummer = Range("roterBereich").Cells(3,1)
```

```
Nummer = Cells(9,9)
```

```
Nummer = Range("startwert")
```

b) Ausgabe von Werten

Startwert: 34

		2				
8			64		76	
69	57	60	65	72	77	
70	58	61	66	73	78	
71	59	62	67	10	79	
		14		75		

Ausgabe

`Tabelle1.Range("roterBereich").Cells(1,1).Value = 8`

`Range("roterBereich").Cells(4,5) = 10`

`Tabelle1.Range("blauerBereich").Cells(1,1).Value = 2`

`Range("blauerBereich").Cells(6,2) = 14`

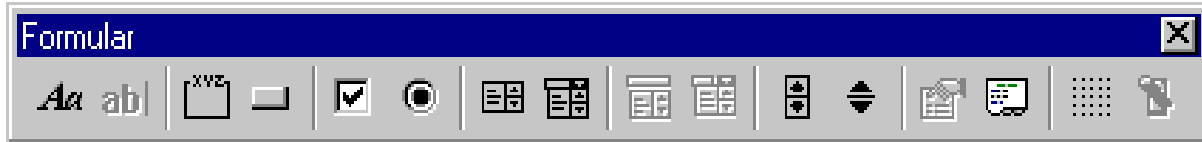
`Tabelle1.Range("startwert") = 34`

`Tabelle1.Cells(1,2) = "Arbeiten mit Range-Objekten"`

Ereignisprozeduren starten die Verarbeitung

Steuerelemente können von der

- (Excel) **Formular**-Symbolleiste oder von der



- **Visual Basic**-Symbolleiste stammen.



⇒ Wir verwenden immer die **Formular**-Symbolleiste.

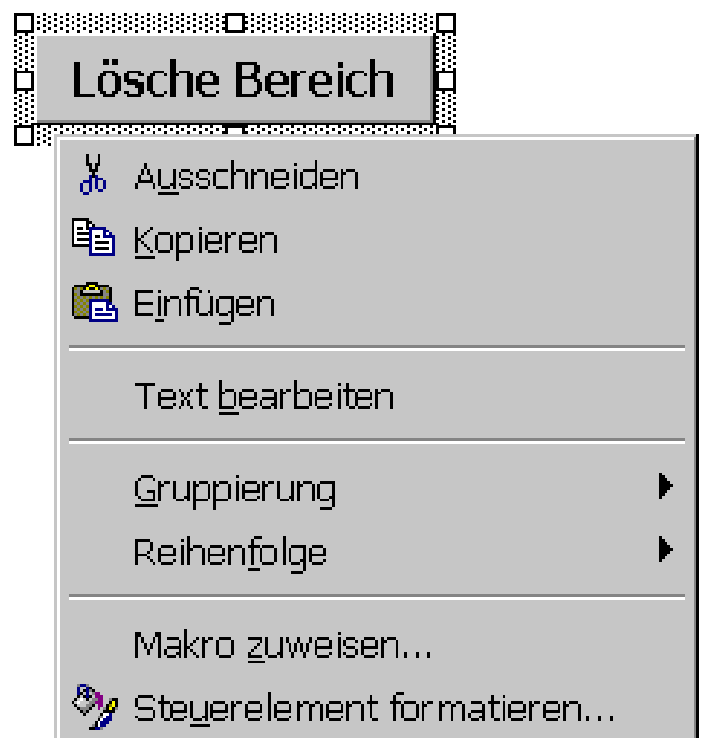
2) **Eigenschaften** festlegen

⇒ *Grösse, Position, Farbe, Schrift, ...*

⇒ Definieren Sie **Namen**, um vom Programm aus auf Zellbereich (und Steuerelemente) zugreifen zu können.

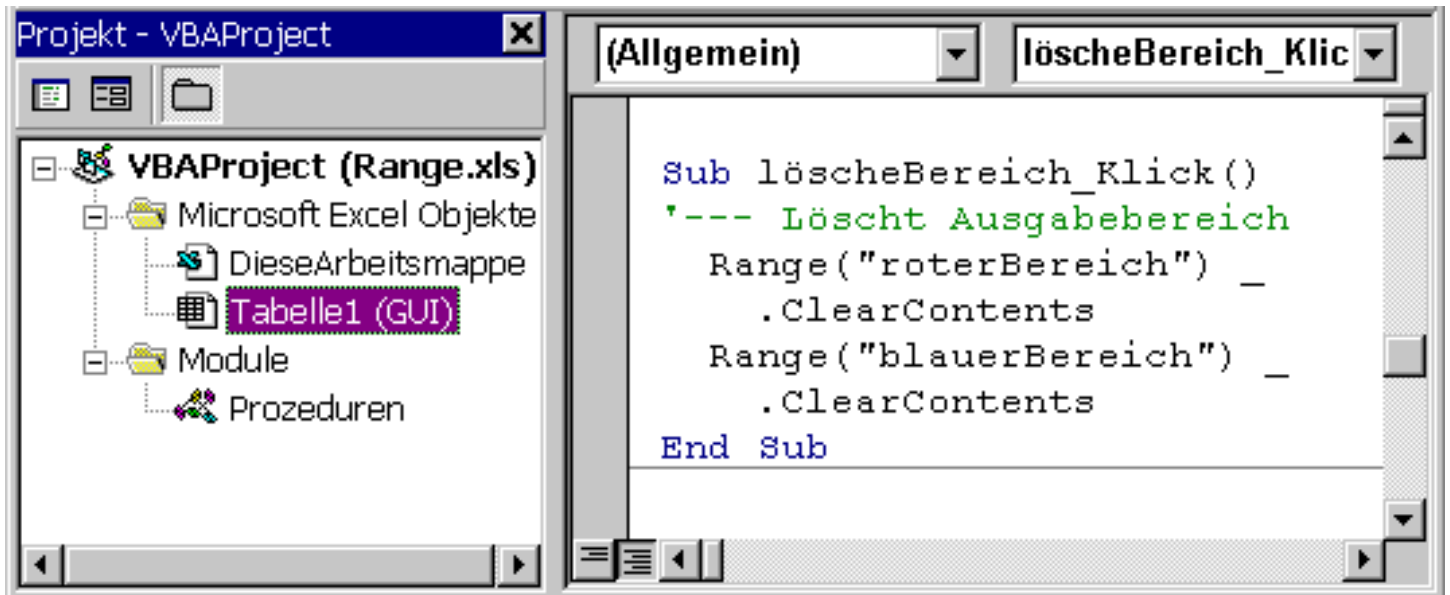
⇒ Weisen Sie den Steuerelementen **Makros** (Ereignisprozeduren) zu, um die **Verarbeitung** starten zu können.

z.B. löscheBereich_Klick



Verhalten am Beispiel Range.xls

3) Verhalten programmieren



a) neues **Modul** erstellen und benennen

z.B. *Prozeduren*

b) **Prozedurkopf** der Ereignisprozedur schreiben

z.B. **Sub** löscheBereich_Klick()
 '--- Löscht den Ausgabebereich.
 |
 End Sub

c) Verhalten im **Prozedurrumpf** programmieren

z.B. **Sub** löscheBereich_Klick()
 '--- Löscht den Ausgabebereich.
 Range("roterBereich").ClearContents
 Range("blauerBereich").ClearContents
 End Sub

wichtige Werkzeuge

Frage: Wie finde ich die passenden Funktionen?

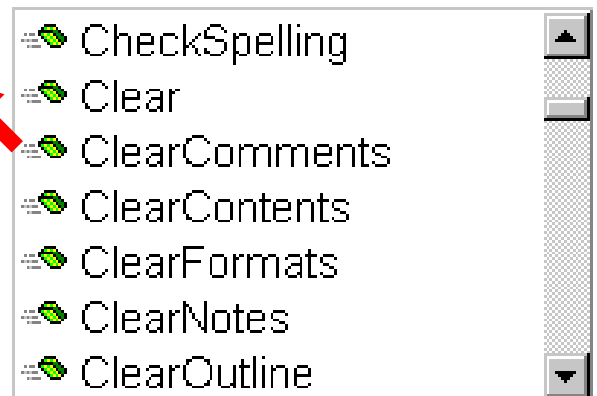
Wie lösche ich beispielsweise den Ausgabebereich?

- Der **Objektbrowser** macht Vorschläge

z.B. `Range("roterBereich").`

Vorschläge: `Clear ?`
 `ClearContents ?`
 `Delete ?`

andere Möglichkeiten?



- **Handbücher** und **Online-Hilfe** erklären Funktionalität und Syntax und geben **Beispiele**

- ✗ `Clear`: **ChartArea-, Legend- oder Range-**Objekte: Löscht das gesamte Objekt.
- ✓ `ClearContents`: Löscht die Formeln aus dem Bereich, jedoch *nicht* die Formatierung.
- ✗ `Delete`: Löscht das Objekt.

- **Bibliotheken** erweitern um zusätzliche Funktionen

- Der **Debugger** veranschaulicht den Ablauf

Debugger am Beispiel Range.xls

4) Programm testen



```
Sub Eingabe_Klick()  
    Dim Nummer As Integer 'Beispielvariable  
  
    Nummer = Range("roterBereich").Cells(1, 3).Value  
    Nummer = Range("roterBereich").Cells(3, 1)  
    Nummer = Cells(9, 9)  
    Nummer = Range("startwert")  
    Nummer = 36  
End Sub
```

Identifizieren Sie die folgenden Begriffe in den abgebildeten Darstellungen. Erklären Sie Funktion und Wirkung.


- | | |
|------------------------|-------------------------------|
| 1. Einzelschritt | 7. Makro ausführen |
| 2. Variableninhalt | 8. Haltepunkt |
| 3. Kennzeichenleiste | 9. Beenden |
| 4. Testen-Symbolleiste | 10. Codefenster |
| 5. Prozedurschritt | 11. Aktuelle Ausführungszeile |
| 6. Entwurfsmodus | 12. Prozedur abschliessen |
| | 13. aktuellen Wert anzeigen |


Nachschlagewerke (Dokumentationen)

- Microsoft Press, *Microsoft Office 97 Visual Basic Programmer's Guide*, Redmond, etc., 1997, Microsoft Corporation
(auch für Office 2000 und Excel 2000)
- Microsoft Excel Visual Basic Hilfe
☺ im Visual Basic Editor: <F1> drücken
☹ "Büroklammer" fragen

Übersichtsdarstellungen zu VBA

- Michael Koffler, *Excel 2000 programmieren*, München, 2000, Addison-Wesley Verlag
- Excel und Visual Basic Hilfe

☺ in Excel: ? /  → Programmierinformationen
Excel VB-Sprachverzeichnis

☺ im VB-Editor: ? /  → VB Konzepte
VB Verfahren
VB Sprachverzeichnis

Programmierkonzepte

- Rod Stephens, *Ready-to-Run Visual Basic Algorithms*, New York, etc., 1998, Wiley Computer Publishing
- Steven Roman, *Concepts of object-oriented programming with Visual Basic*, New York, 1998, etc., Springer

Beispiel zur Online-Hilfe: Excel-Objekthierarchie

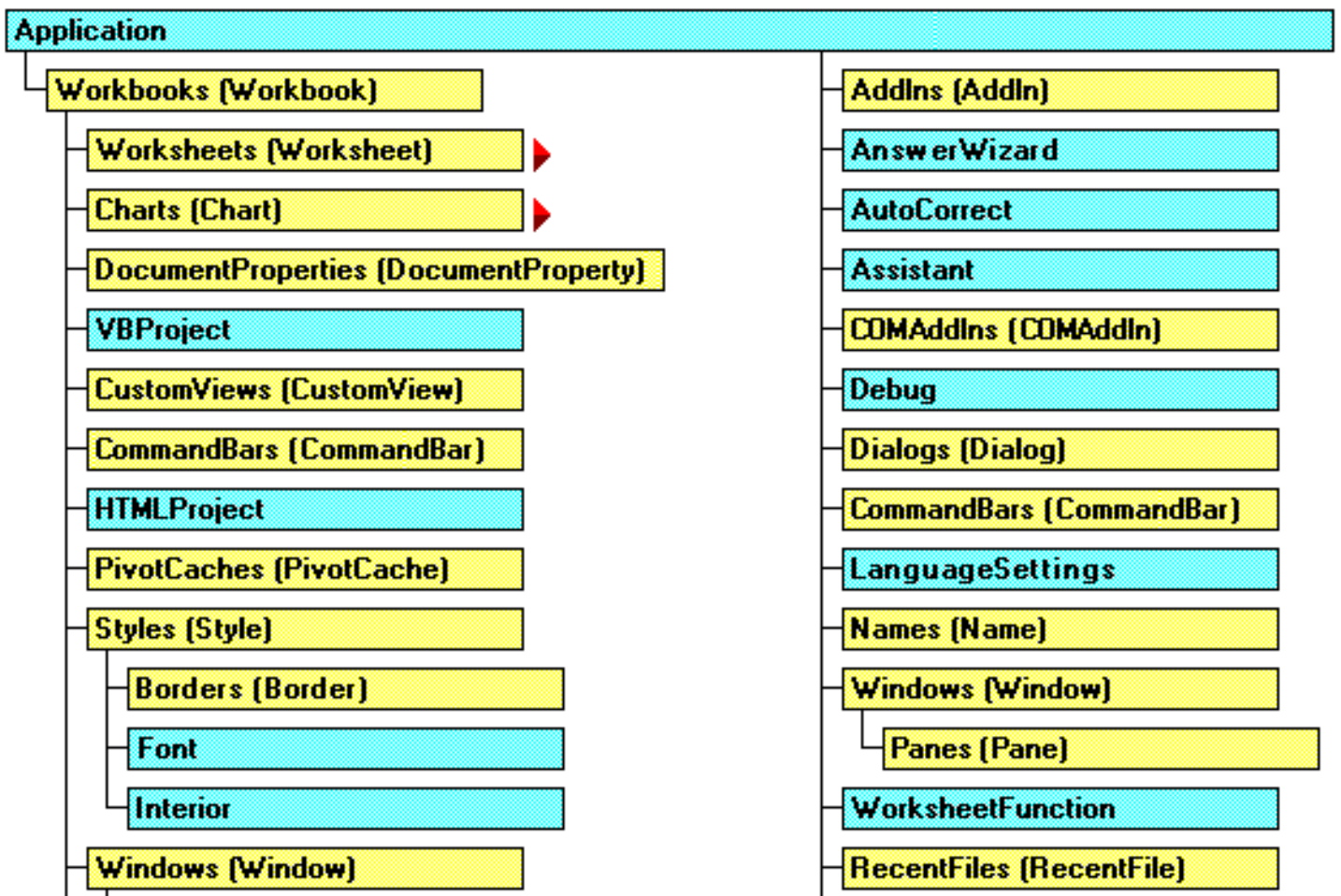
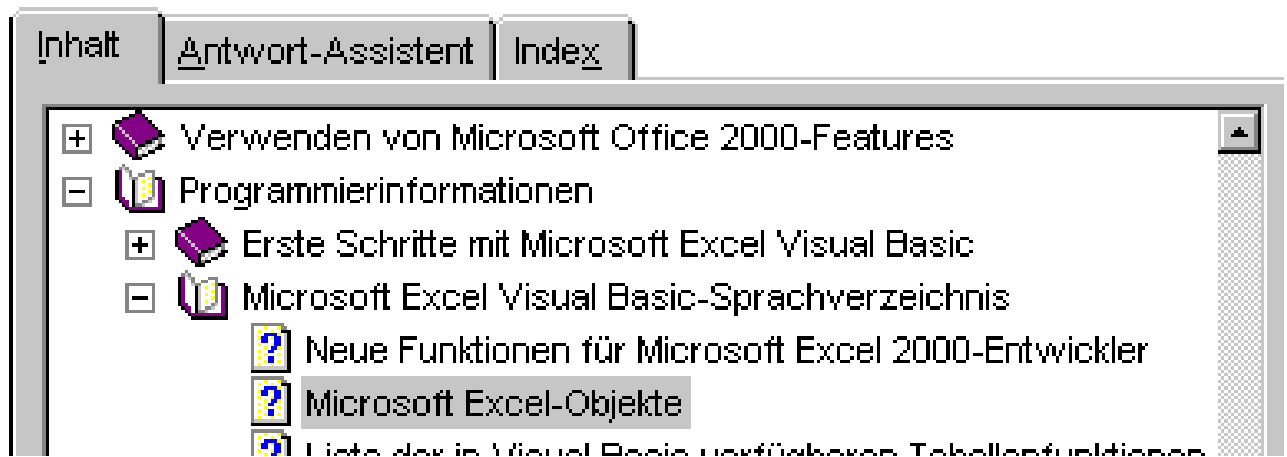
Frage: Wie verschaffe ich mir eine Übersicht?



Excel



MS Excel-Hilfe



Objektmodelle - Programmieren mit Objekten

Ein **Objektmodell** beschreibt die Objekte einer Anwendung und ihre Beziehungen untereinander.

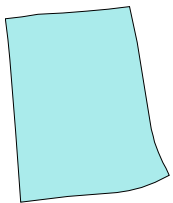
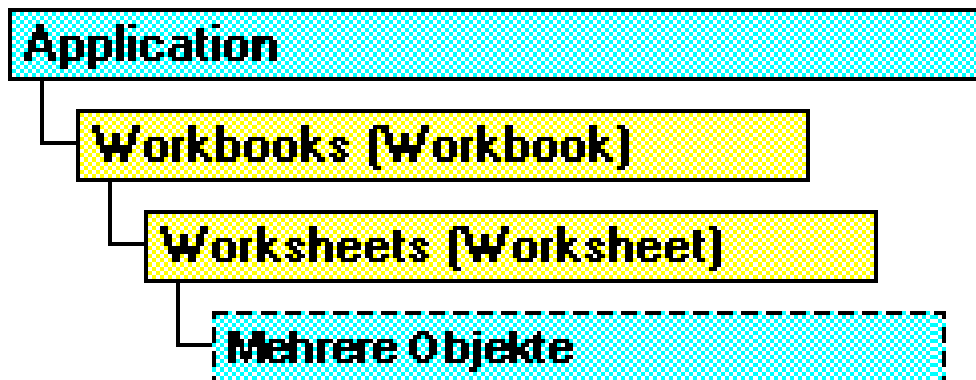
Ein **Objekt** ist eine Abstraktion eines Gegenstandes oder eines Konzeptes.

Als **Abstraktion** bezeichnet man die selektive Auswahl bestimmter Aspekte eines Gegenstandes oder eines Problems, mit dem Ziel, wichtige Aspekte zu isolieren und unwichtige zu ignorieren.

Eine **Objektklasse** ist eine Beschreibung einer Gruppe von Objekten mit gleichen Eigenschaften und gleichem Verhalten. Den *Zustand* beschreibt man durch eine Menge von Daten, das *Verhalten* durch eine Menge von Prozeduren.

Ein einzelnes Objekt, das man aufgrund einer Objektklasse erstellt, nennt man **Instanz** der Klasse.

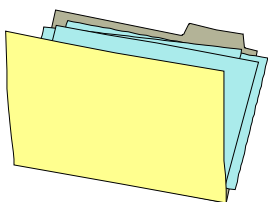
Objekthierarchie: **Objekt** und **Auflistung**



nur **Objekt**

Bsp.: Application
Range

- Ein Objekt ist ein konkretes Teil der laufenden Anwendung, das
- Inhalt und Funktionalität (Daten und Code) zusammenfasst.
- Die Hilfe beschreibt nur die Funktionalität (Objektklasse)



Auflistung und **Objekt**

Bsp.: Workbooks (Workbook)
Worksheets (Worksheet)

- Eine Auflistung ist ein Objekt, das
- andere Objekte enthält.
- typische Methoden: **Add, Delete, Item**
- typische Eigenschaften: **Count, Parent**
- (Zu einem gelben Kästchen finden Sie *zwei* Hilfetexte)

Zugriff auf Objekte einer Auflistung

Startwert: 34

	12				
34			64		42
35	57	60	65	38	43
36	58	61	66	39	44
37	59	62	67	40	45
		14		41	

Zugriff auf *ein* Objekt

Syntax:

Auflistung.Item(Index)

Auflistung.Item(Name)

Auflistung(Index)

Auflistung(Name)

Beispiel: nummeriere roten Zellbereich

```
Nummer = Application _  
    .Workbooks("Range.xls") _  
    .Worksheets("GUI") _  
    .Range("roterBereich") _  
    .Cells(2,3) _  
    .Value
```

Zugriff auf Objekte einer Auflistung

Startwert: 34

	12				
34			64		42
35	57	60	65	38	43
36	58	61	66	39	44
37	59	62	67	40	45
		14		41	

Zugriff auf *alle* Objekte

Syntax:

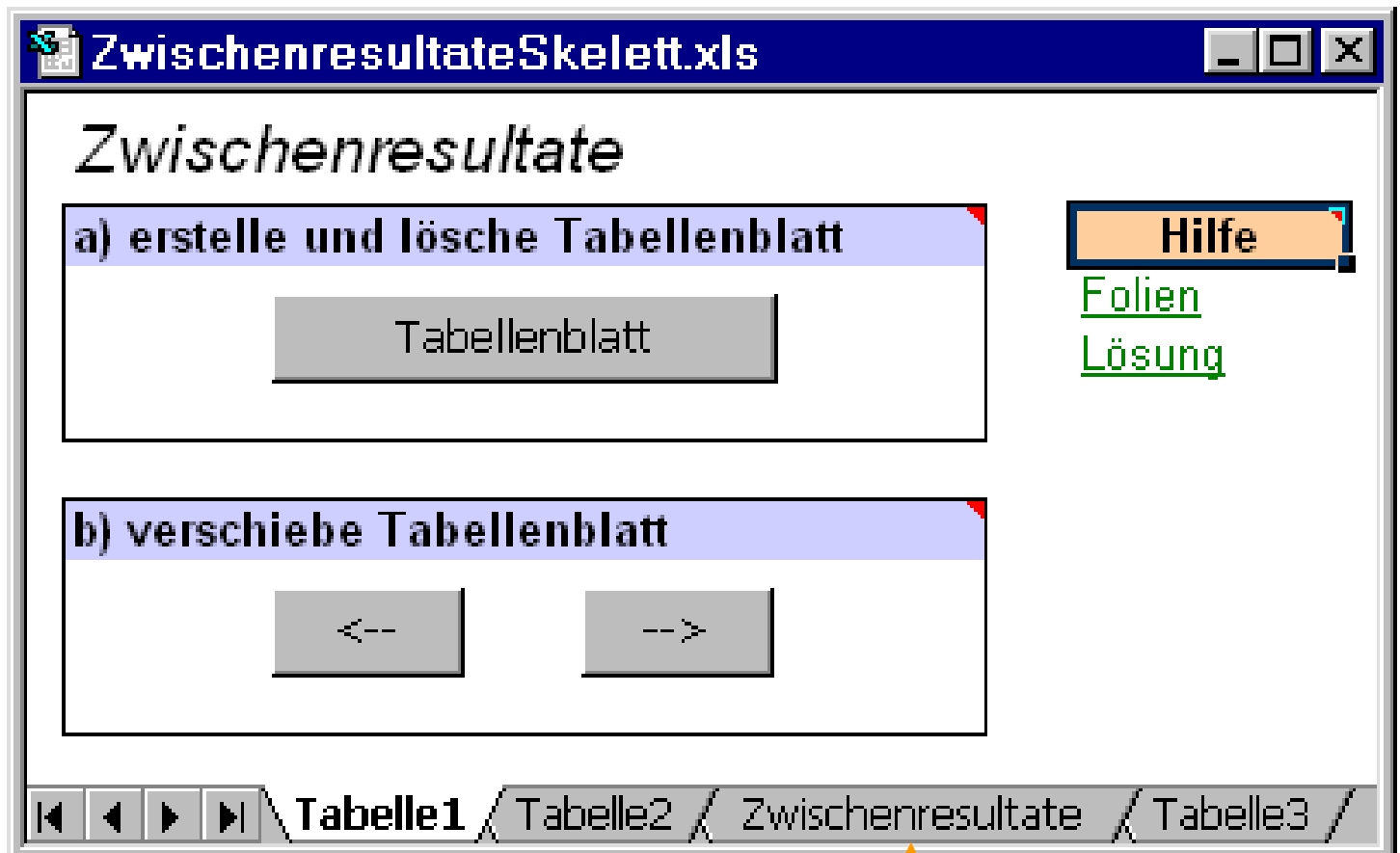
```
For Each Element In Auflistung
    <Anweisungen>
[Exit For]
    <Anweisungen>
Next [Element]
```

Beispiel: nummeriere roten Zellbereich

```
Sub nummeriereRot_Klick()
    Dim Zelle As Range           'ein Element
    Dim Nummer As Integer        'Verwaltung der Nummern
    Nummer = Range("startwert").Value
    For Each Zelle In Range("roterBereich").Cells
        Zelle.Value = Nummer
        Nummer = Nummer + 1
    Next Zelle
End Sub
```

Auflistungen am Beispiel Zwischenresultate.xls

Benutzerschnittstelle



Aufgabe

a) Schreiben Sie ein Programm, ...

- das ein neues Tabellenblatt "Zwischenresultate" in die aktuelle Arbeitsmappe *einfügt*, falls dieses Tabellenblatt noch nicht existiert, bzw.
- es aus der Arbeitsmappe *entfernt*, falls es bereits existiert.

b) Schreiben Sie ein Programm, das die Position des Tabellenblatts "Zwischenresultate"

- nach *links* bzw.
- nach *rechts verschiebt*.

Arbeiten mit Arbeitsblättern

Hilfetext zum Worksheets-Objekt (Auflistung)

⇒ Verwenden der Worksheets-Auflistung

⇒ Liste der Eigenschaften und Methoden

⇒ Unterschied zur Sheets-Auflistung

⇒ Beispiel zur Move-Methode

```
Worksheets.Move After:=Sheets(Sheets.Count)
```

⇒ Beispiel zur Add-Methode

```
Worksheets.Add Count:=2, Before:=Sheets(1)
```

Prüfe, ob “Zwischenresultate” bereits existiert

```
Dim Tabellenblatt As Worksheet
```

```
Dim BlattGefunden As Boolean
```

```
BlattGefunden = False
```

```
For Each Tabellenblatt In Worksheets
```

```
    If Tabellenblatt.Name = "Zwischenresultate" Then
```

```
        BlattGefunden = True
```

```
    End If
```

```
Next Tabellenblatt
```

Arbeitsblatt einfügen und entfernen

Tabellenblatt einfügen bzw. entfernen

Syntax:

- `Worksheets.Add([After], [Before])`
 - Erstellt neues Arbeitsblatt und gibt es zurück.
 - Wenn Before und After nicht angegeben werden, wird das neue Blatt vor dem aktiven Blatt eingefügt.
- `Worksheets.Item(Index/Name).Delete`
 - Ruft die Delete-Methode der Worksheet-Klasse auf.

Beispiel:

```
If BlattGefunden = True Then  
    Worksheets("Zwischenresultate").Delete  
Else  
    Worksheets.Add.Name = "Zwischenresultate"  
End If
```

Arbeitsblatt verschieben

Index: 1 2 3 4

\ **Tabelle1** / Tabelle2 / Zwischenresultate / Tabelle3 /

Tabellenblatt verschieben

Syntax:

- Worksheets.Item(Index/Name).**Move**([After], [Before])
 - Ruft die Move-Methode der Worksheet-Klasse auf.
- Worksheets.Item(Index/Name).**Index**
 - Gibt die "Position" eines Objektes in einer Auflistung zurück.
- Worksheets.**Count**
 - Gibt die Anzahl Objekte in einer Auflistung zurück.

Beispiel: Verschiebe nach rechts

```
Dim Tabellenblatt As Worksheet
For Each Tabellenblatt In Worksheets
    If Tabellenblatt.Name = "Zwischenresultate" Then
        If Tabellenblatt.Index < Worksheets.Count Then
            Tabellenblatt.Move _
                After:=Worksheets(Tabellenblatt.Index + 1)
        End If
    End If
Next Tabellenblatt
```

benannte Argumente

Beispiel

Syntax der **Move**-Methode: `Ausdruck.Move(Before, After)`

Before : optionales Argument

After : optionales Argument

unbenannte Argumente

Bei der Übergabe von Argumenten *ohne* Angabe der Namen muss die richtige *Reihenfolge* eingehalten werden.

Beispielaufruf:

`Tabellenblatt.Move 4`


vor oder hinter 4

benannte Argumente

Bei der Angabe ihrer Namen können Argumente in *beliebiger* Reihenfolge übergeben werden.

Beispielaufrufe:

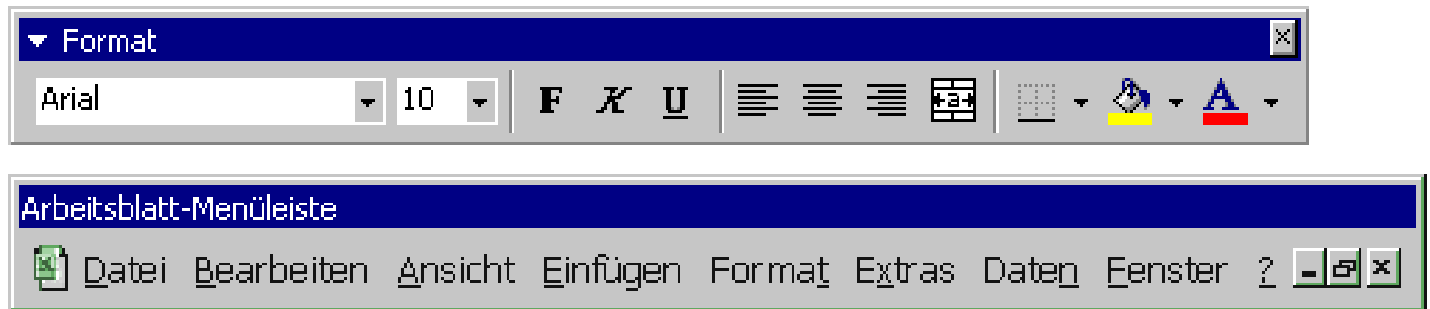
`Tabellenblatt.Move Before:=6`

`Tabellenblatt.Move After:=5`

Übungsaufgabe kleineHelfer.xls

Menüs und Menüpunkte

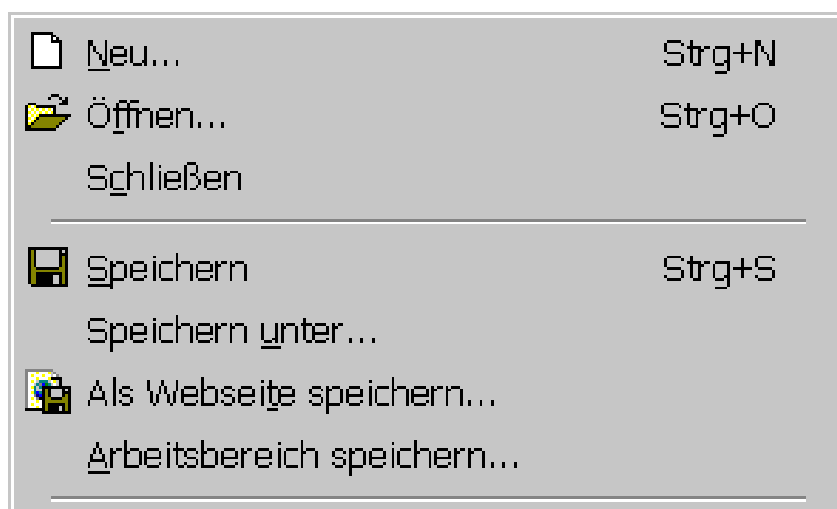
1. Eine Anwendung besitzt **Menüs**



2. Ein Menü besitzt **Menüpunkte**



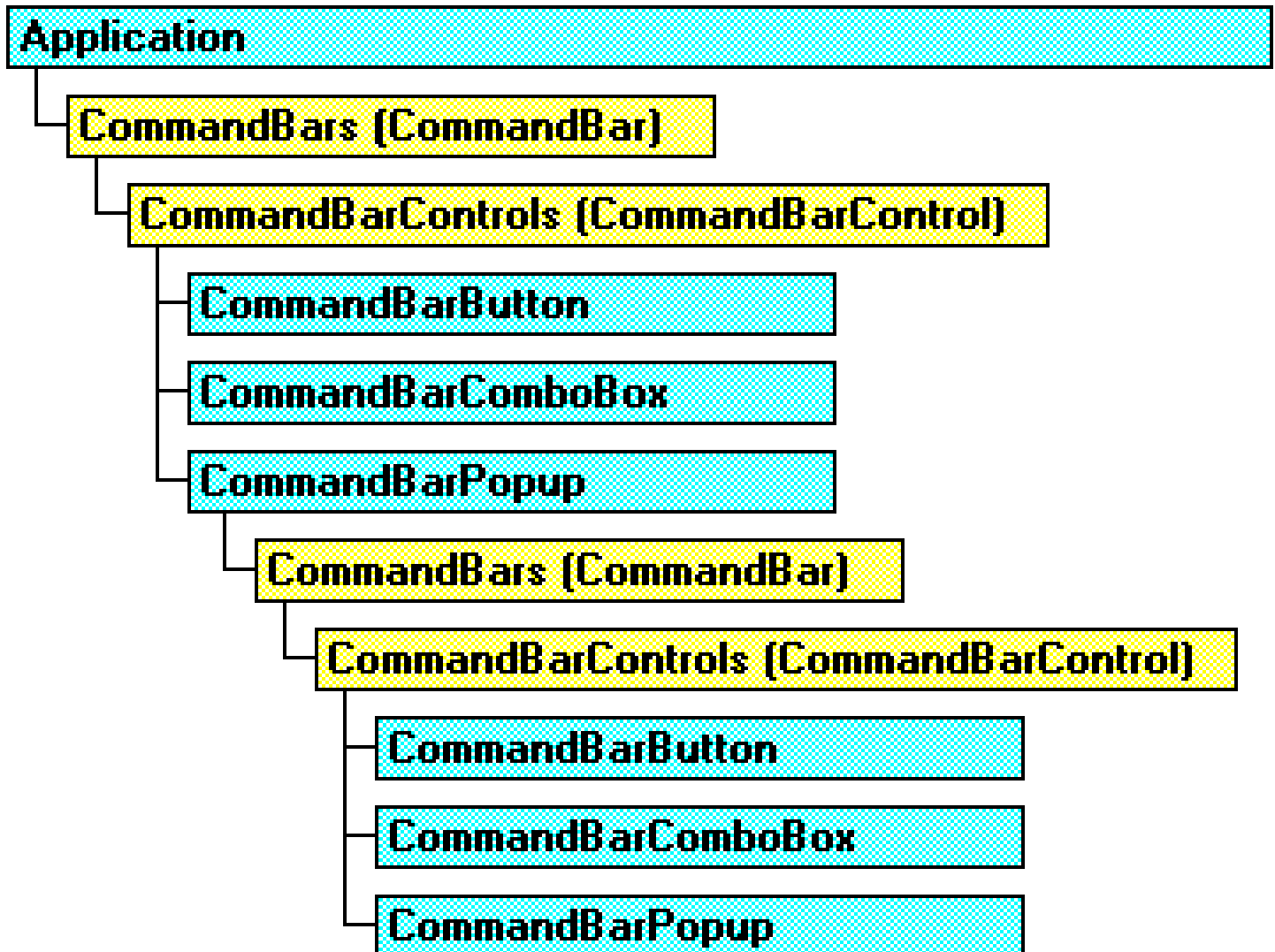
3. Ein Menüpunkt führt Funktionen aus oder besitzt **Untermenüs**



4. Ein Untermenü besitzt **Menüpunkte**

usw.

Objektmodell für Menüs und Menüpunkte



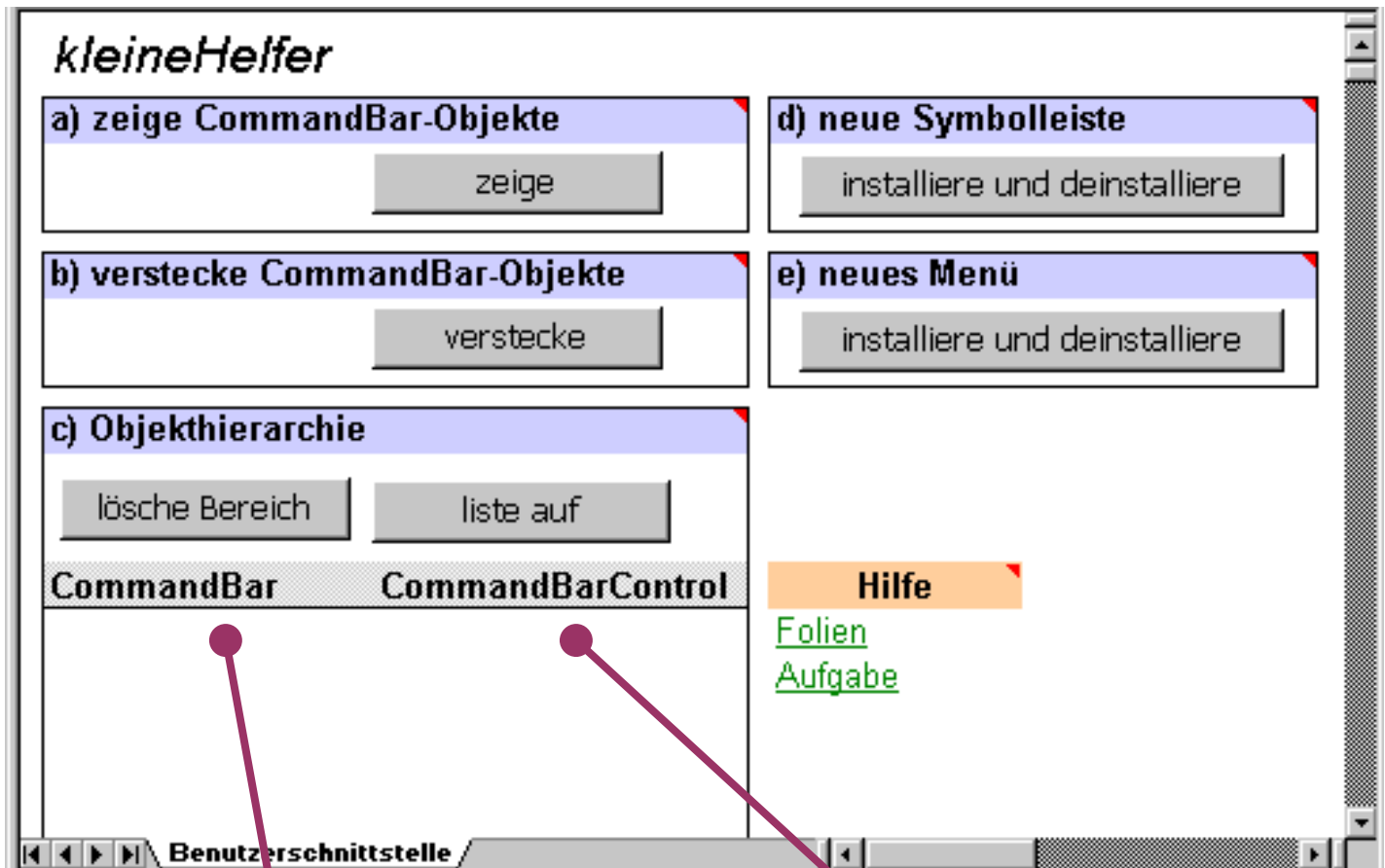
CommandBars

Eine Auflistung von **CommandBar**-Objekten, die die Befehlsleisten (Menüs) darstellen.

CommandBarControls

Eine Auflistung von **CommandBarControl**-Objekten, die die Befehlsleisten-Steuererelemente (Menüpunkte) auf einer Befehlsleiste (Menü) darstellen.

Benutzerschnittstelle



Range(„commandBar“)

Range(„commandBarControl“)

Lernziele

- ⇒ Excel-Objekthierarchie
- ⇒ Benutzung der Online-Hilfe
- ⇒ Auflistung
- ⇒ For Each ... Next-Schleife
- ⇒ Ein- und Ausgabe auf Tabellenblatt