

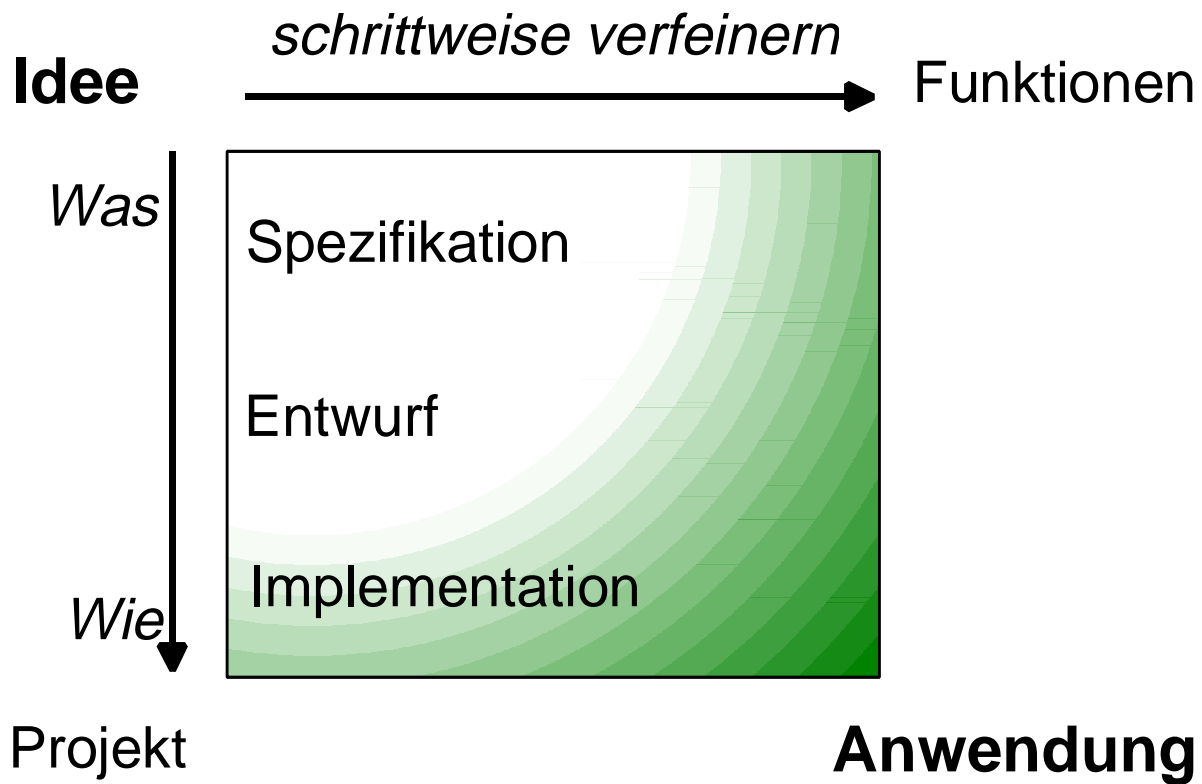
Andreas Born

# Anwendungsentwicklung

## 2. Teil

Programmieren mit Objekten  
unter MS Excel und VBA

# Anwendungsentwicklung



## andere Blickrichtungen

- Eingabe - Verarbeitung - Ausgabe
- Daten - Ablauf

---

# Einordnung

---

## Deklarativ Anwendungen entwickeln

- 📖 Tabellenkalkulation ✓
- 📖 Elementare Was-Wenn-Analyse ✓
- 📖 Lineare Optimierung ✓

## Prozedural Anwendungen entwickeln

- 📖 Datentypen und Ablaufstrukturen ✓
- 📖 Benutzerschnittstelle ✓
- 📖 Datenfeld ✓
- ⇒ Wo.1. Elemente von Programmiersprachen (Repetition)
- ⇒ Wo.2. Werkzeuge, Objekthierarchie, Auflistung
- ⇒ Wo.3. Objekte, Objektvariablen, Objektklassen
- ⇒ Wo.4. benutzerdefinierte Objektklassen
- ⇒ Wo.5. Datenstrukturen, verkettete Liste, Binärbaum
- ⇒ Wo.6. Algorithmen, Baumtraversierungen
- ⇒ *Repetitorium, Musterklausur*

## Datenbankanwendungen entwickeln

- 📖 Dateiverwaltung
- 📖 Datenentwurf
- 📖 Datenbankverwaltung
- 📖 Anwendungsentwicklung

---

# Organisation

---

## Vorlesungen und Übungen

- Vorlesung Montag (oder Donnerstag)
  - Sprechstunden
- Übungen
  - verteilt auf die ganze Woche
  - Tutoren
  - WISIS
  - Vorbereitung

## Veranstaltungsunterlagen

- Merkblätter → Web, Anschlagsbrett 3.Stock
- Druckerkärtchen → Sekretariat
- Skript auf CD → Buch-CD
- Beispiele → Buch-CD, Web
- Übungen → Buch-CD, Web, Laborserver
- Foliensammlung → Buch-CD, Web

## WebSite

<http://www.wwz.unibas.ch/wi/>

<http://www.wwz.unibas.ch/wi/lehre/i2/mbi2.html>

---

# 1. Woche

---

## Vorlesung: Elemente von Programmiersprachen

Entwicklungswerkzeuge

Modularisierung

Daten

Konstante, Variable

Datentyp

ausführbare Anweisungen

Prozedur

Sequenz

Entscheidung

Wiederholung

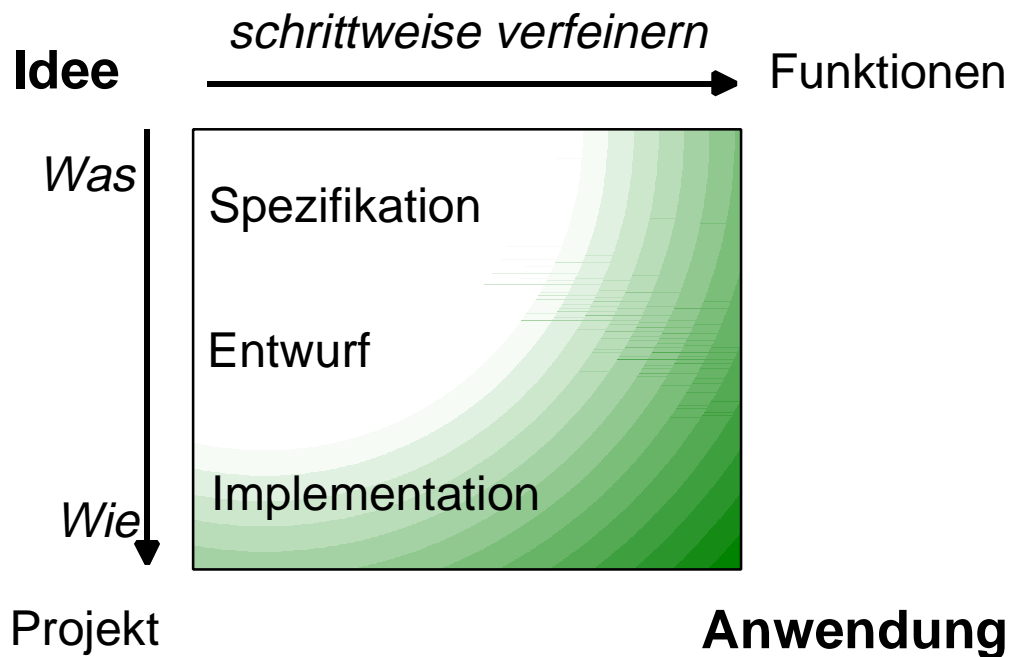
Verschachtelung

Testen

## Übung: Turtle

Koordinatensystem

# Entwicklungswerkzeuge



## Werkzeuge eines Entwicklers

- a) Tabellenblatt und Formulare
- b) Programmeditor
- c) Projektextplorer
- d) Textverarbeitung
- e) Debugger
- f) Papier und Bleistift
- g) Compiler bzw. Interpreter

# Verfeinern durch Modularisieren

## Verfeinerungsstufen

Projekt

Modul

Prozedur

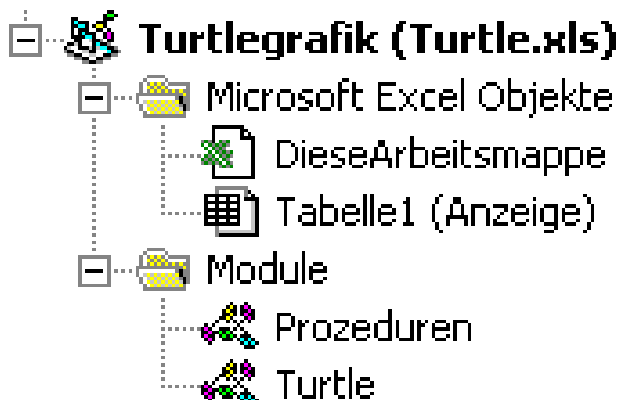
elementare Anweisung

## Modul

Ein **Modul** ist ein entwurfs- oder programmiersprachlicher Baustein, der Aufgaben zusammenfasst, die eng verbunden sind

- Aufgabe
- Modularten

## Beispiele



## 2 Klassen

2.1 Objektbeschreibungen.....

2.2 Instanzen.....

2.2.1 Instanzen erstellen.....

2.2.2 Beispiel Telefonverzeichnis

---

# Aufbau eines Code-Moduls

---

## 1. Anweisungen an den Übersetzer

z.B.

*Option Explicit*

## 2. Vereinbarungsabschnitt

- Variablen und Konstanten
- öffentlich und privat

z.B.:

```
'Private Konstanten  
Private Const Pi2 As Single = 6.2831856  
  
'Private Variablen  
Private X As Integer
```

## 3. Prozeduren

- Sub-Prozeduren, Funktionen und Ereignisprozeduren
- öffentlich und privat

z.B.:

```
'Öffentliche Prozeduren  
Public Sub Start()  
  
    lösche  
    With Worksheets("Anzeige")  
        ...  
  
End Sub
```



# Vereinbarungsanweisungen

Eine **Variable** (Veränderliche) ist ein Name eines Speicherbereiches mit änderbarem Inhalt.

Eine **benannte Konstante** ist ein Name eines Speicherbereiches, dessen Inhalt sich während des Programmablaufes nicht ändert.

Eine Variable (Konstante) **vereinbaren** heisst einem Speicherplatz einen *Namen*, einen *Datentyp*, einen *Gültigkeitsbereich* und einen *Anfangswert* zuweisen

## Ort

- Vereinbarungabschnitt eines Moduls
- Prozedurkopf oder Prozedurrumpf

## Syntax

{**Public** | **Private** | **Dim**} [**Const**] Name **As** Typ [=Wert]

## Begriffe

- Gültigkeitsbereich
- Lebensdauer
- Initialisieren
- Operationen
- Wertebereich

# Datentypen

Ein **Datentyp** legt den *Speicherplatzbedarf*, den *Wertebereich* und die *Operationen* einer Variablen oder Konstanten fest.

## Datentypen

Name	Speicher	Wertebereich	Operationen
<b>einfach</b>			
Integer	2 Bytes	-32'768 bis 32'767	+, -, *, /, Mod
Boolean	2 Bytes	True oder False	Not, And, Or
Single	4 Bytes	-10E38 bis 10E38	+, -, *, /, Mod
<b>zusammengesetzt</b>			
String	1 Byte pro Zeichen	Text aus ANSI-Zeichen (64K)	&, +
Datenfeld	(elementabhängig)		
Satz	(elementabhängig)		

## Begriffe

- einfach - zusammengesetzt
- systemdefiniert - benutzerdefiniert

# Zusammengesetzte Datentypen

## Datenfeld

Ein **Datenfeld** ist eine Folge von Zellen des *gleichen* Datentyps, auf die unter dem gleichen Namen, aber verschiedenen *Indizes* zugegriffen werden kann.

Vereinbarung:

```
Private Läufer(1 To 5) As String
```

Zugriff bzw. Zuweisung:

```
Läufer(1) = "Anna"
```

```
Läufer(2) = "Beat"
```

```
Läufer(3) = "David"
```

```
...
```

## Satz

Ein **Satz** ist eine Struktur aus gleichen oder *verschiedenen* Datentypen.

Vereinbarung:

```
Type tPerson
```

```
    Name As String
```

```
    Rang As Integer
```

```
End Type
```

```
Private Läuferin As tPerson
```

Zugriff bzw. Zuweisung:

```
Läuferin.Name = "Anna"
```

```
Läuferin.Rang = 2
```

# Prozeduren

Eine **Prozedur** (Unterprogramm) ist ein Baustein, der aus Programmteilen aufgerufen werden kann.

## Ort

- nach Vereinbarungsabschnitt eines Moduls

## Syntax

```
[Public | Private] {Sub | Function} Name( _  
    [[ByVal | ByRef] Argument1 As Typ, ...] ) [As Typ]  
    <Vereinbarungsanweisungen>  
    <ausführbare Anweisungen>  
End {Sub | Function}
```

## Begriffe

- Gültigkeitsbereich
- Sub-Prozedur, Funktion, Ereignisprozedur
- Argument
- Rückgabewert
- lokale Variable
- ausführbare Anweisung

Eine Prozedur ist ein Modul

# Anweisung

## Vereinbarungsanweisungen

### ausführbare Anweisungen

#### vordefiniert

einfach	Calculate
parametrisiert	AddLine <b>X1, Y1, X2, Y2</b>
mit Rückgabewert	Len( <b>Text</b> )

#### benutzerdefiniert

einfach	QuadratFesterLänge
parametrisiert	Quadrat <b>X, Y, Seitenlänge</b>
mit Rückgabewert	einWortWeniger( <b>Text</b> )

#### Zuweisungsanweisungen

schreiben (links von '=')	<b>Länge</b> = Len(Text)
lesen (rechts von '=')	neueLänge = <b>Länge</b> + 10

#### Entscheidungsanweisungen

If-Anweisung	<b>If</b> Länge = 0 <b>Then</b>
--------------	---------------------------------

#### Wiederholungsanweisungen

Zählschleife	<b>For</b> i = 1 <b>To</b> 5 ... <b>Next</b>
mit Ausführbedingung	<b>While</b> Len(Text) > 0 ... <b>Loop</b>

Ausführbare Anweisungen stehen immer innerhalb einer Prozedur.

## andere

#### Kommentar

'x-Koordinate

#### Einstellungen

Option Explicit

#### Zeile fortsetzen

—

# Operatoren

## Zuweisung

<linke Seite> = <rechte Seite>

### 1. werte rechte Seite aus

*Operatorvorrang (arithmetisch)*

<b>^</b>	Potenzierung
<b>-</b>	Negation
<b>*, /</b>	Multiplikation und Division
<b>\</b>	Ganzzahldivision
<b>Mod</b>	Restwert
<b>+, -</b>	Addition und Subtraktion
<b>&amp;</b>	Zeichenverkettung

### 2. weise Resultat der linken Seite zu

*Datentypen müssen sich vertragen!*

## Vergleich

<linke Seite> {=, <>, <, >, <=, >=, **Is**} <rechte Seite>

### werte linke und rechte Seite aus und vergleiche Resultate

*Operatorvorrang*

arithmetische Operatoren: **^**, (**\***, **/**), **\**, **Mod**, (**+**, **-**), **&**

Vergleichsoperatoren: **=**, **<>**, **<**, **>**, **<=**, **>=**, **Is**

logische Operatoren: **Not**, **And**, **Or**

*Datentypen müssen sich vertragen!*

Runde **Klammern** ändern den Vorrang.

# Argumente

Das **Übergeben** eines Arguments (als Wert) bei einem Prozeduraufruf entspricht einer **Zuweisung** an eine Variable.

## Beispiel

```
Public Sub Quadrat(ByVal x As Integer, _  
                  ByVal y As Integer)  
    ...
```

```
Public Sub Test()  
    ...  
    A = 100  
    Quadrat A, 200
```

Entspricht Zuweisungen: **x** = **A**, **y** = **200**

## Begriffe

- als Wert (ByVal) - als Adresse (ByRef)
- Positionsargumente - benannte Argumente
- voreingestellte Argumentwerte

# Entscheidungsanweisungen

## If-Anweisung

If Bedingung Then

*<Anweisungsblock>*

[Else]

*<Anweisungsblock>*

End If

Ein Gruppe zusammengehörender Anweisungen  
heisst **Anweisungsblock**.

- einzelne Anweisung
- Prozedur
- Sequenz
- Entscheidungsanweisungen
- Wiederholungsanweisungen

Anweisungsblöcke werden in VBA meist mit **End**  
(oder **Loop** oder **Next**) abgeschlossen.



# Schleifenanweisungen

## Zählschleife

```
For Zähler = Anfang To Ende  
    <Anweisungsblock>  
[Exit For]  
    <Anweisungsblock>  
Next [Zähler]
```

## Schleife mit Ausführbedingung

```
<Initialisierung>  
Do While Bedingung  
    <Anweisungsblock>  
[Exit Do]  
    <Anweisungsblock>  
Loop
```

## Schleife mit Abbruchbedingung

```
<Initialisierung>  
Do  
    <Anweisungsblock>  
[Exit Do]  
    <Anweisungsblock>  
Loop While Bedingung
```

## Begriffe

- Initialisierung
- Schleifenbedingung
- Schleifenkörper

# Schleifenanweisungen

## Wann benutzt man welchen Schleifentyp?

Schleife	For ... To ... Next	Do While ... Loop	Do ... Loop While
Kriterium			
Abbruch- bedingung	<b>vor</b> Schleifenkörper	<b>vor</b> Schleifenkörper	<b>nach</b> Schleifenkörper
Abbruch- kriterium	Zählerstand	True	True
Anzahl Durchläufe	$\geq 0$	$\geq 0$	$\geq 1$
Anzahl vorher bekannt?	ja	nein	nein
Initialisierung	implizit	explizit	explizit
Beispiel	Datenfeld	Datei	Datei

Die **Rekursion** ist eine weitere Form der Wiederholung.

# Verschachtelung ausführbarer Anweisungen

Die rekursive Definition von Anweisungsblöcken führt zu **Verschachtelungen**.

- z.B.: Die If-Anweisung ist ein Anweisungsblock.

**If** Bedingung **Then**

*<Anweisungsblock>*

**[Else]**

*<Anweisungsblock>*

**End If**

- z.B.: Schleife und Entscheidungen

Anzahl = 0

**Do** While Len(Text) > 0

**If** erstesZeichen(Text) = "a" **Then**

Anzahl = Anzahl + 1

**End If**

entferneErstesZeichen(Text)

**Loop**

Die Entscheidungs- oder Schleifenanweisung, die *zuletzt* geöffnet wurde, muss *zuerst* wieder geschlossen werden.

# Verschachtelung von Datenstrukturen

Durch verschachteln benutzerdefinierte Datentypen entstehen komplexe **Datenstrukturen**.

- z.B.: Satz und Datenfeld (->Tabelle)

```
Type tPerson                                'Spalten
    Name As String
    Rang As Integer
End Type
Dim Läufer(1 To 5) As tPerson              'Zeilen
```

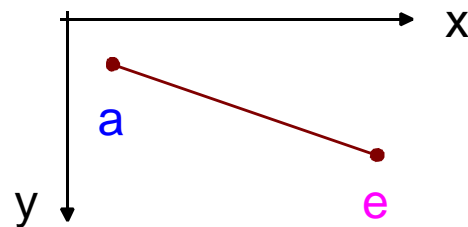
Läufer	Name	Rang
1	Anna	2
2	Beat	4
3	Carmen	1
4	David	3
5	Elisabeth	5

# Koordinatensystem

## Excel

In Excel zeichnet man Linien durch Angabe eines **Anfangs-** und eines **Endpunktes** bezüglich eines Koordinatensystems (**absolut**).

```
Ausgabeblatt.Shapes _  
    .AddLine(Xa, Ya, Xe, Ye).Visible = True
```



## Turtle-Grafik

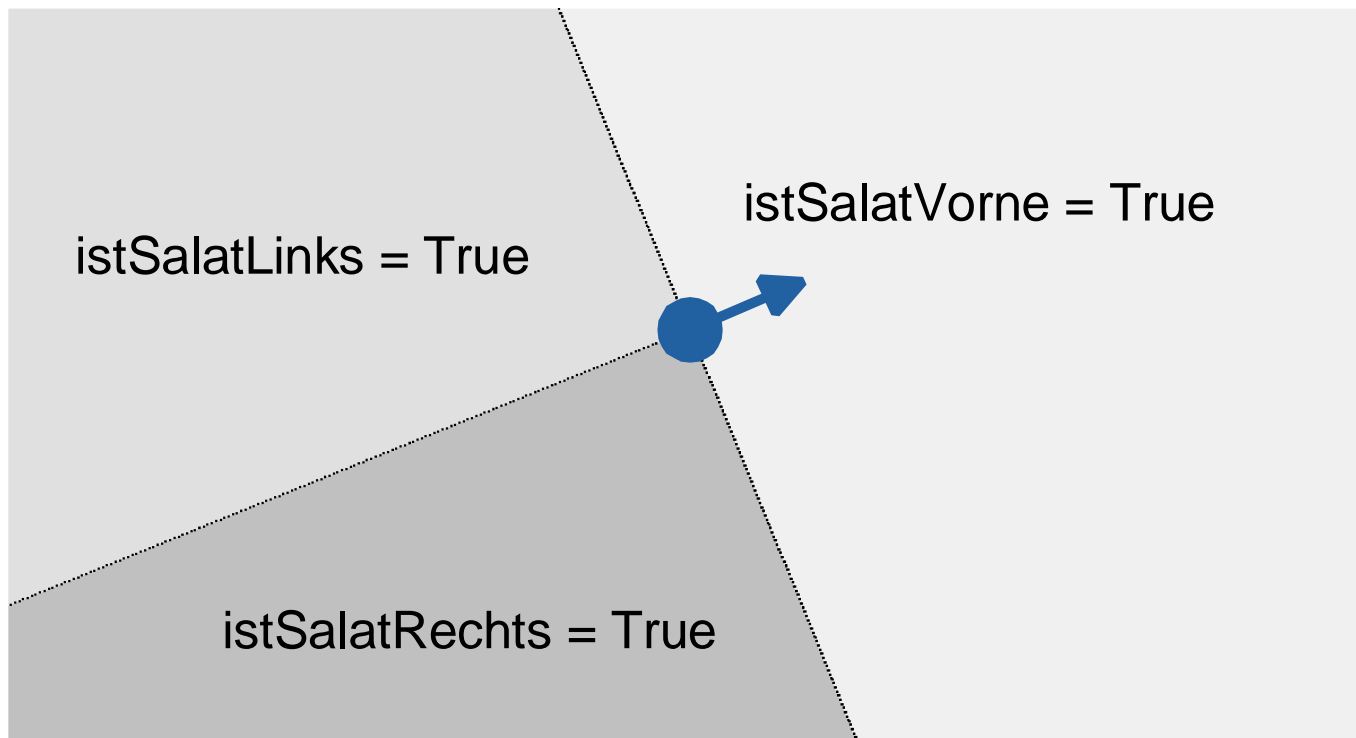
In der Turtle-Grafik zeichnet man Linien durch Angabe einer **Richtung** und einer **Länge relativ** zur aktuellen Position und Richtung.

- Start
- geheVorwärts <Schritte>
- dreheLinks <Winkel>
- dreheRechts <Winkel>



## zusätzliche Prozeduren

- `istSalatVorne`
- `istSalatLinks`
- `istSalatRechts`
- `istSalatGefunden`



---

## 2. Woche

---

### Vorlesung: Objekte und Objektmodelle

#### Entwicklungswerkzeuge

- Programmieren mit VBA und Excel

  - Benutzerschnittstelle gestalten

    - Eingabe entgegennehmen

    - Ausgabe anzeigen

    - Verarbeitung anregen

#### Literatur und Hilfe

- Objekthierarchie von Excel

#### Objektmodelle - Programmieren mit Objekten

- Objekt und Auflistung

  - Zugriff auf Objekte einer Auflistung

  - Auflistung am Beispiel

    - Arbeiten mit Arbeitsblättern

      - Arbeitsblätter einfügen und entfernen

      - Arbeitsblätter verschieben

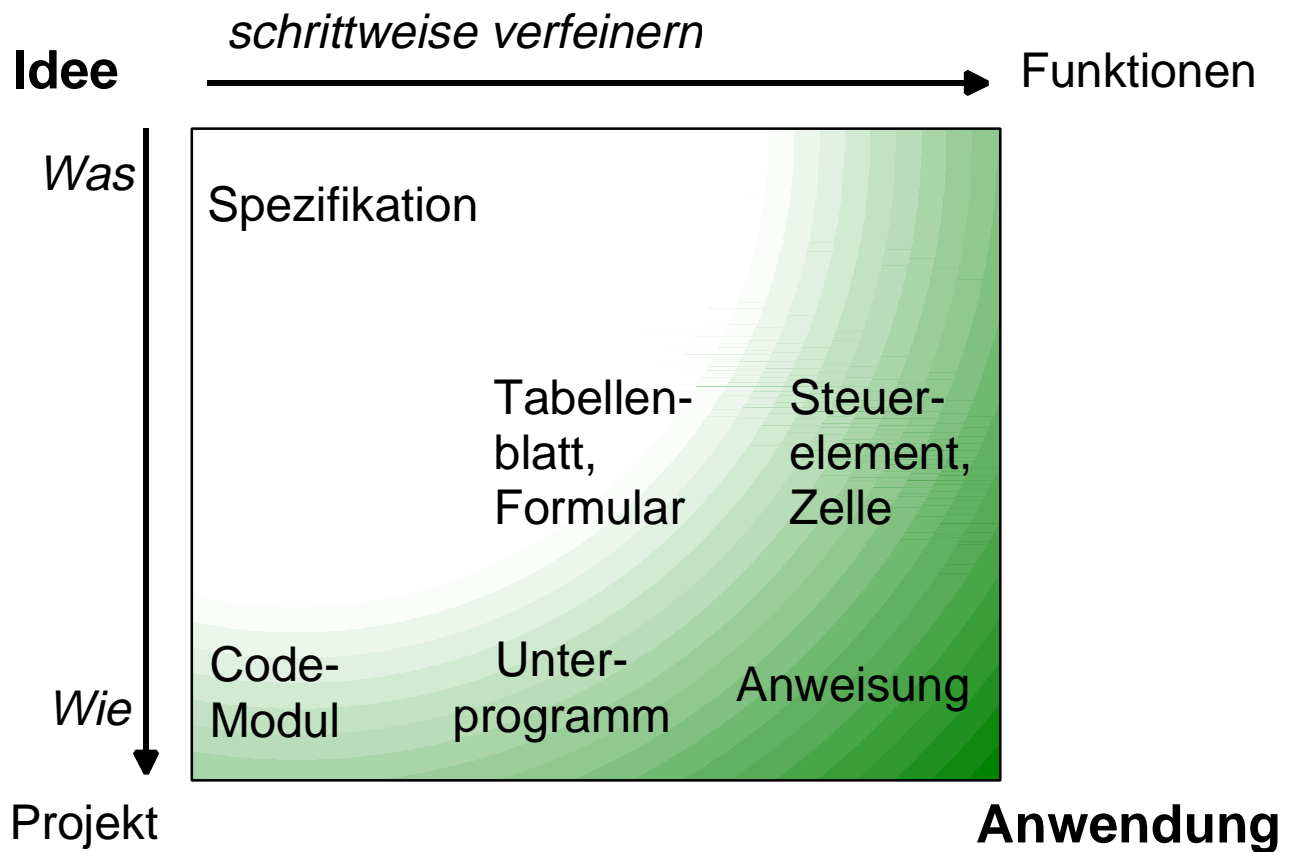
### Übung: kleineHelfer

Excel-Objekthierarchie, Benutzung der Online-Hilfe

Auflistung, For Each ... Next-Schleife

Ein- und Ausgabe auf Tabellenblatt

# Entwicklungswerkzeuge



## Werkzeuge eines Entwicklers

- a) Tabellenblatt und Formulare
- b) Programmeditor
- c) Projektextplorer
- d) Textverarbeitung
- e) Debugger
- f) Papier und Bleistift
- g) Compiler bzw. Interpreter



---

# Excel und VBA als Entwicklungsumgebung

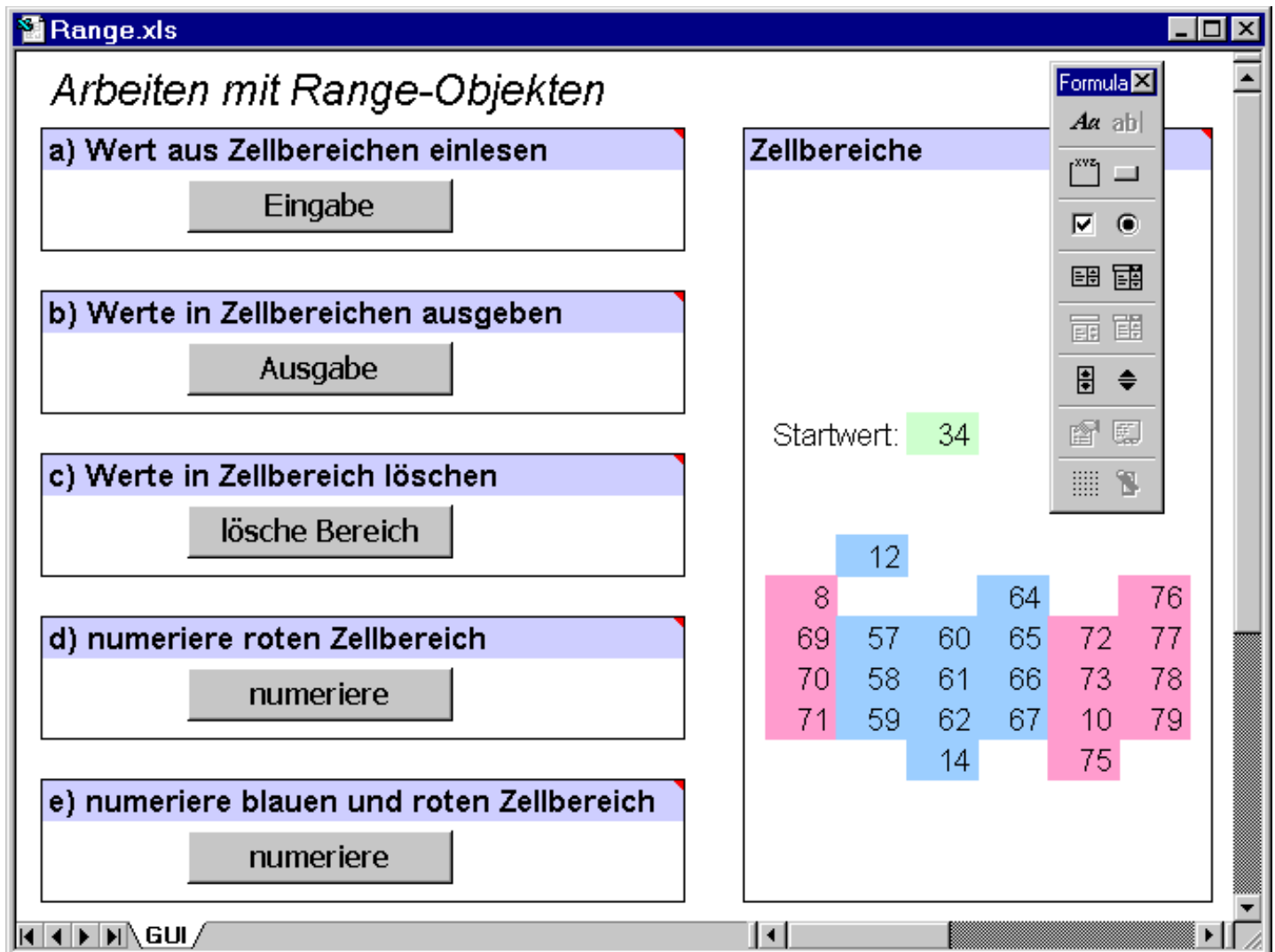
---

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



**E**ingabe...      **Z**ellbereiche (benannt)

**V**erarbeitung...      **S**teuerelemente (v.a. Schaltflächen)

**A**usgabe...      **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
```

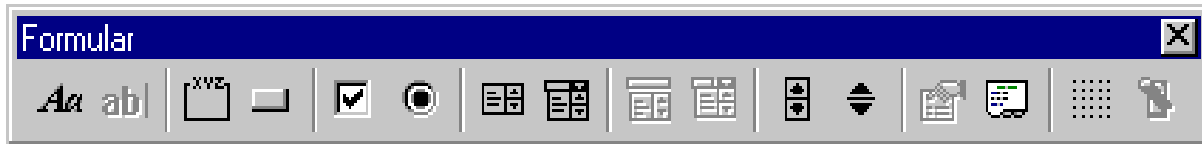
```
Tabelle1.Range("startwert") = 34
```

```
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.

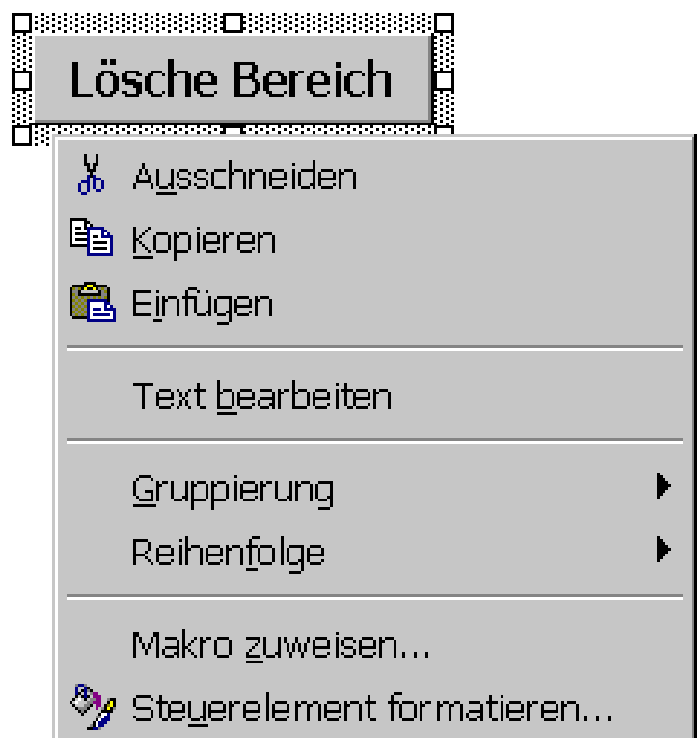
## 2) **Eigenschaften** festlegen

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

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

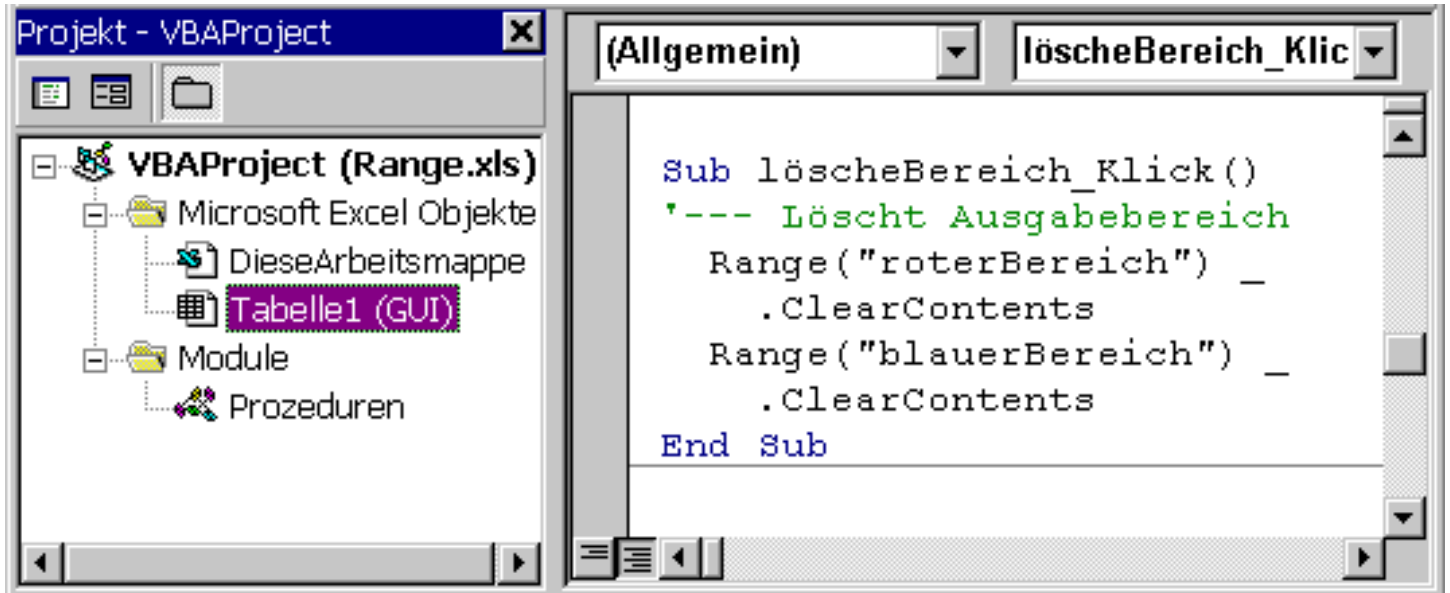
⇒ Weisen Sie den Steuerelementen **Makros** (Ereignisprozeduren) zu, um die **Verarbeitung** starten zu können.

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?

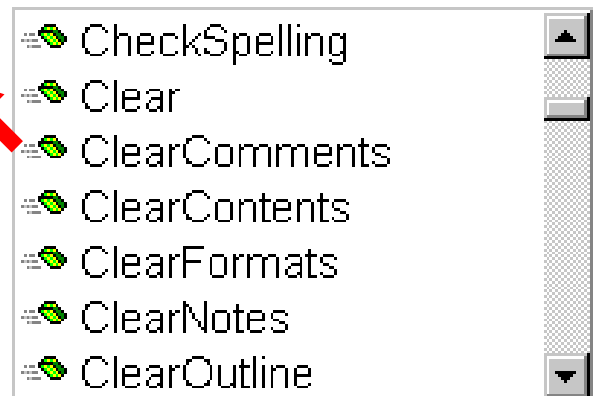
Wie lösche ich beispielsweise den Ausgabebereich?

- 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 von Drittanbietern selber erstellt

- 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
- Alistair McMonnies, *Visual Basic - An Object Oriented Approach*, Harlow, England, 2001, Pearson Education Limited

# Beispiel zur Online-Hilfe: Excel-Objekthierarchie

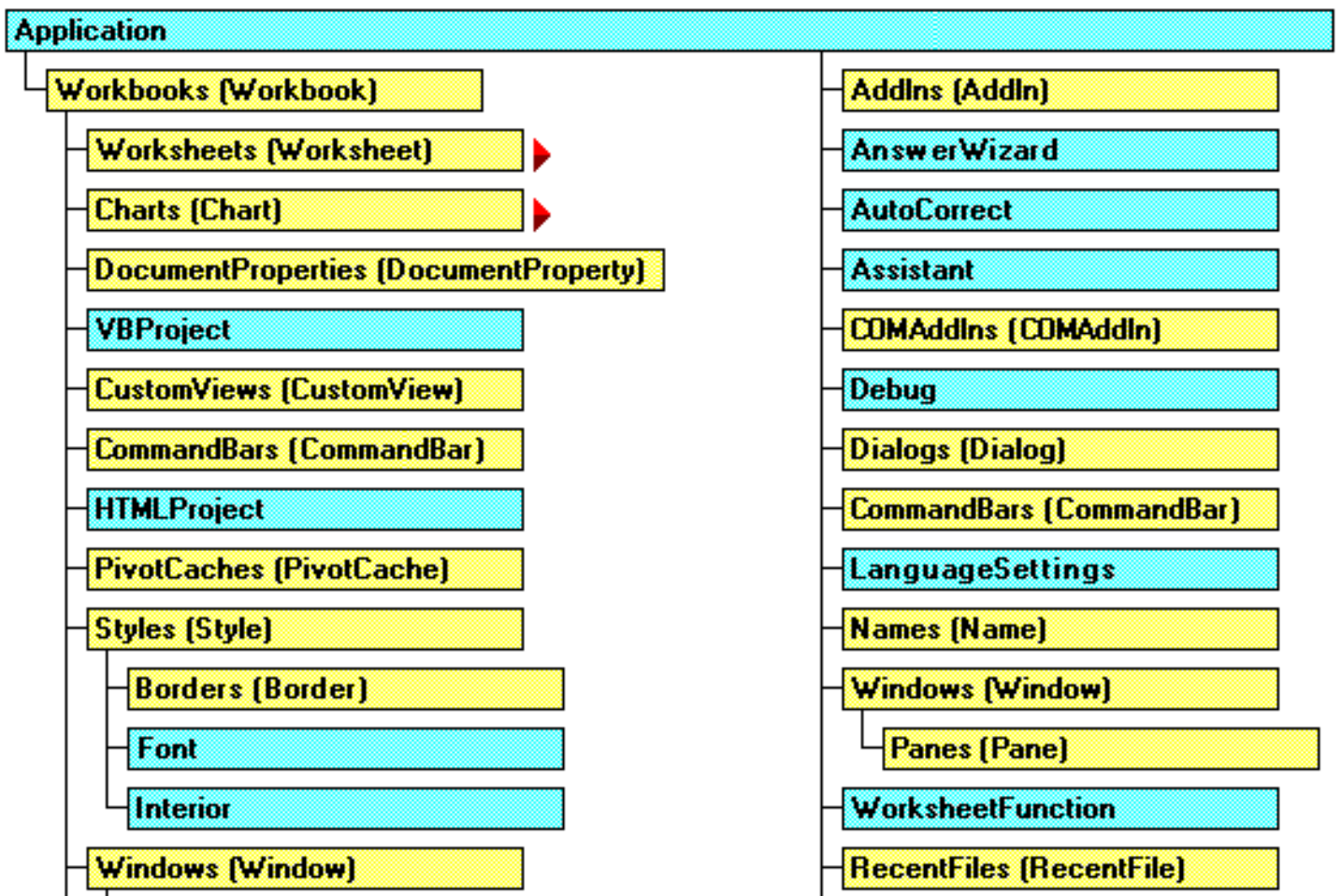
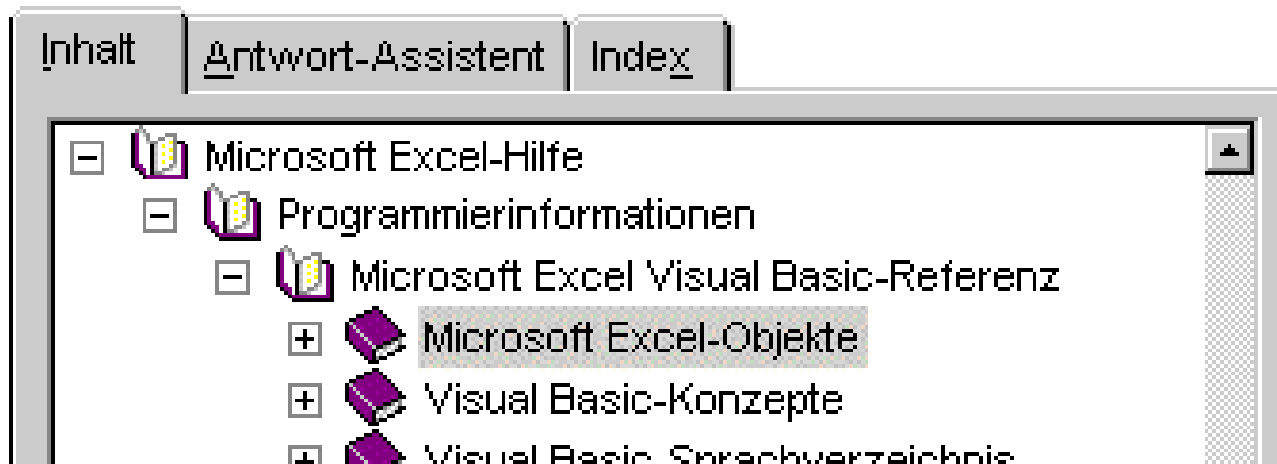
**Frage:** Wie verschaffe ich mir eine Übersicht?



Excel



MS Excel-Hilfe



# Objektmodelle - Programmieren mit Objekten

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

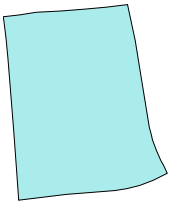
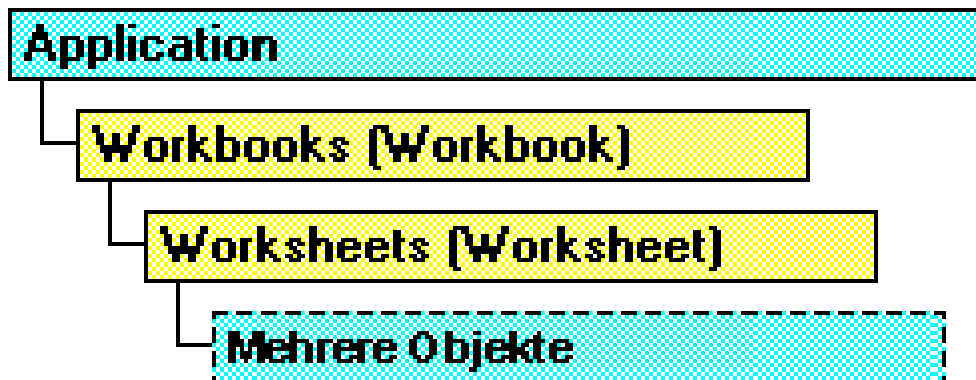
Ein **Objekt** ist eine Abstraktion eines Gegenstandes oder eines Konzeptes.

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.

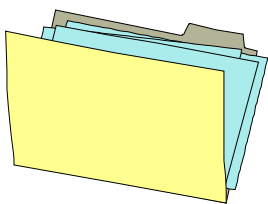
# Objekthierarchie: **Objekt** und **Auflistung**



## 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:

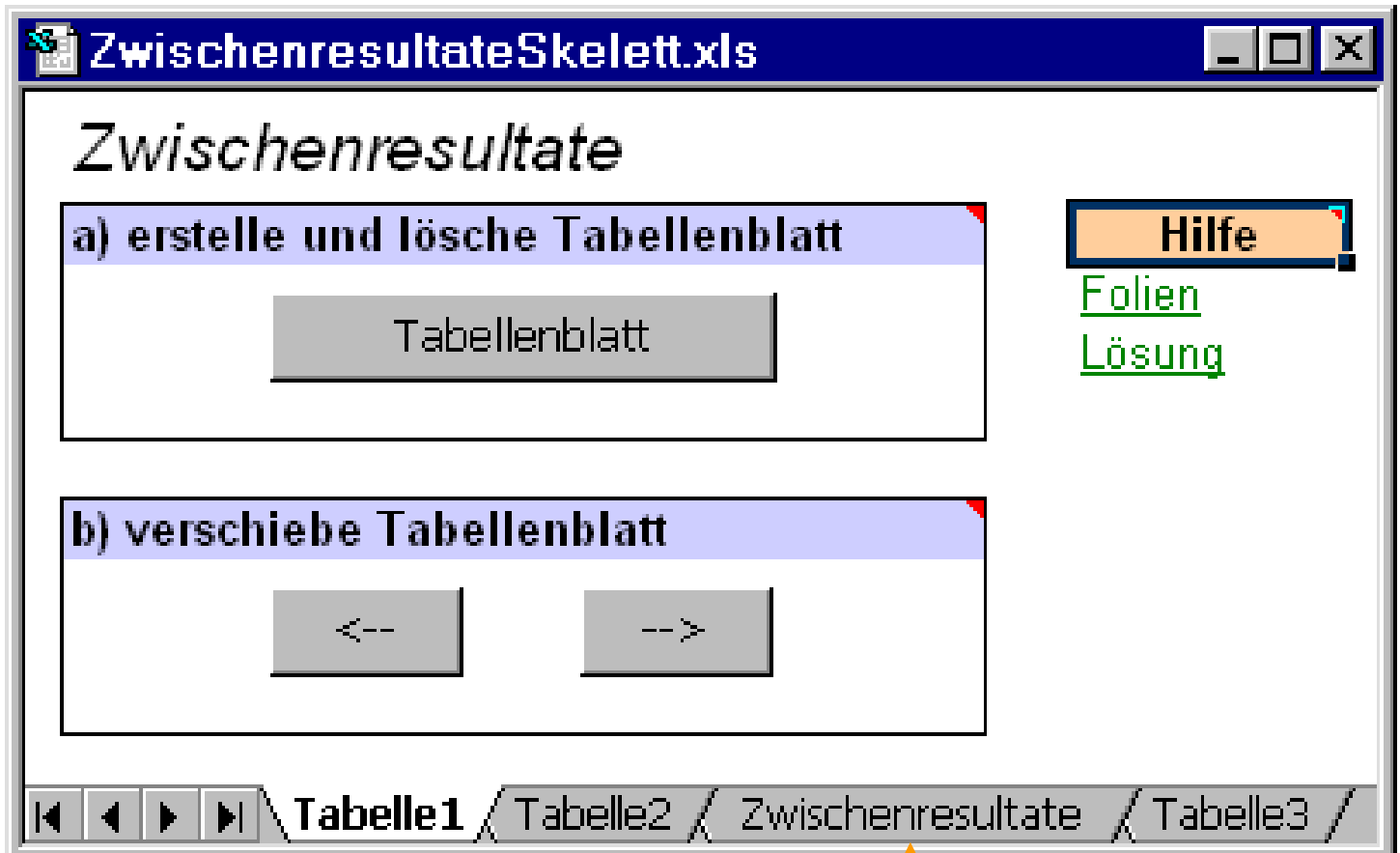
```
For Each Element In Auflistung
    <Anweisungen>
[Exit For]
    <Anweisungen>
Next [Element]
```

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.

### Beispielaufufe:

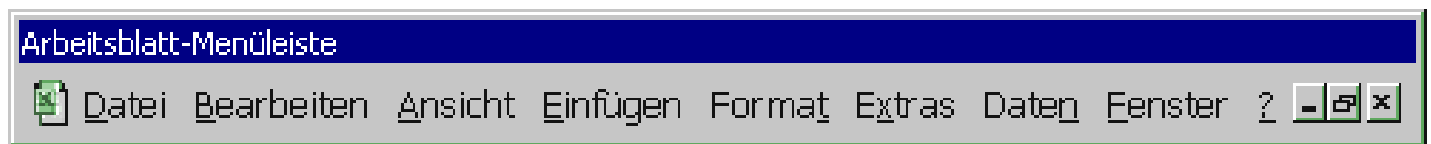
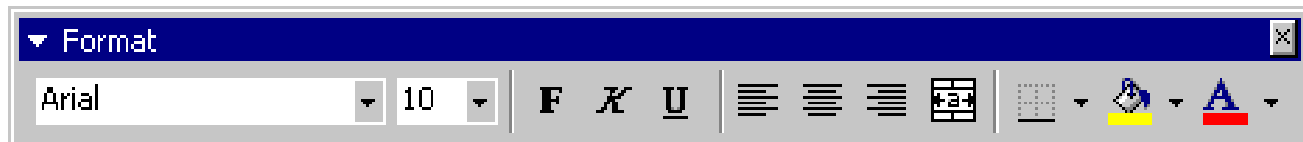
`Tabellenblatt.Move Before:=6`

`Tabellenblatt.Move After:=5`

# Übungsaufgabe kleineHelfer.xls

## Menüs und Menüpunkte

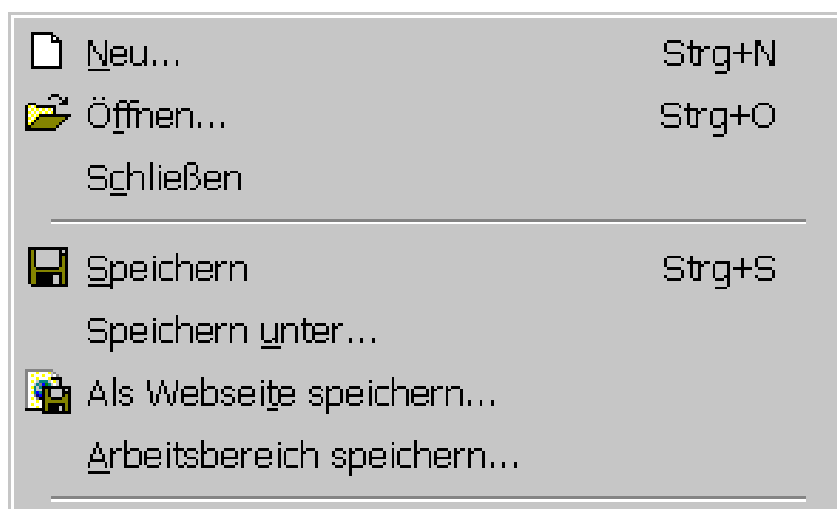
### 1. Eine Anwendung besitzt **Menüs**



### 2. Ein Menü besitzt **Menüpunkte**



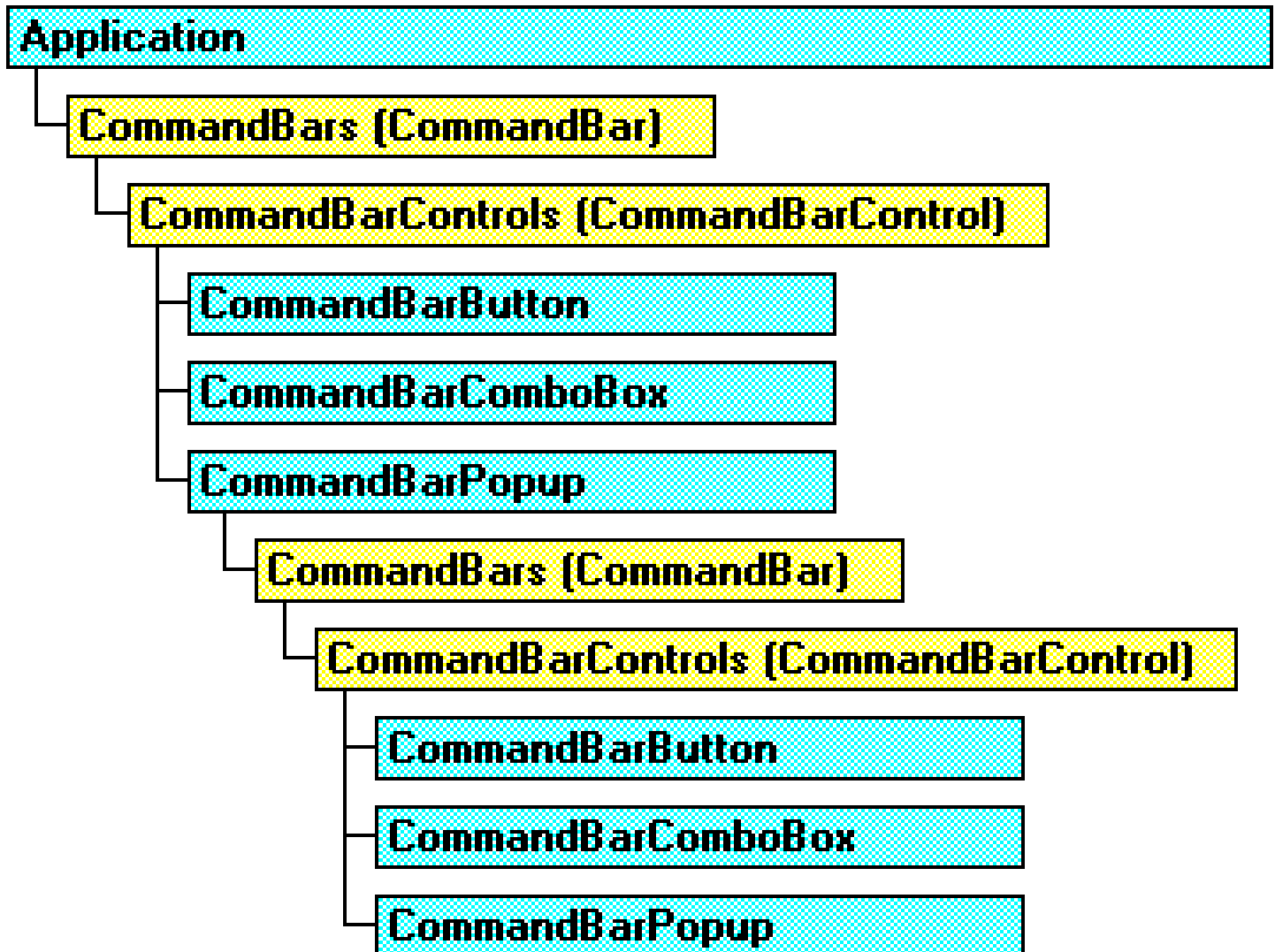
### 3. Ein Menüpunkt führt Funktionen aus oder besitzt **Untermenüs**



### 4. Ein Untermenü besitzt **Menüpunkte**

usw.

# Objektmodell für Menüs und Menüpunkte



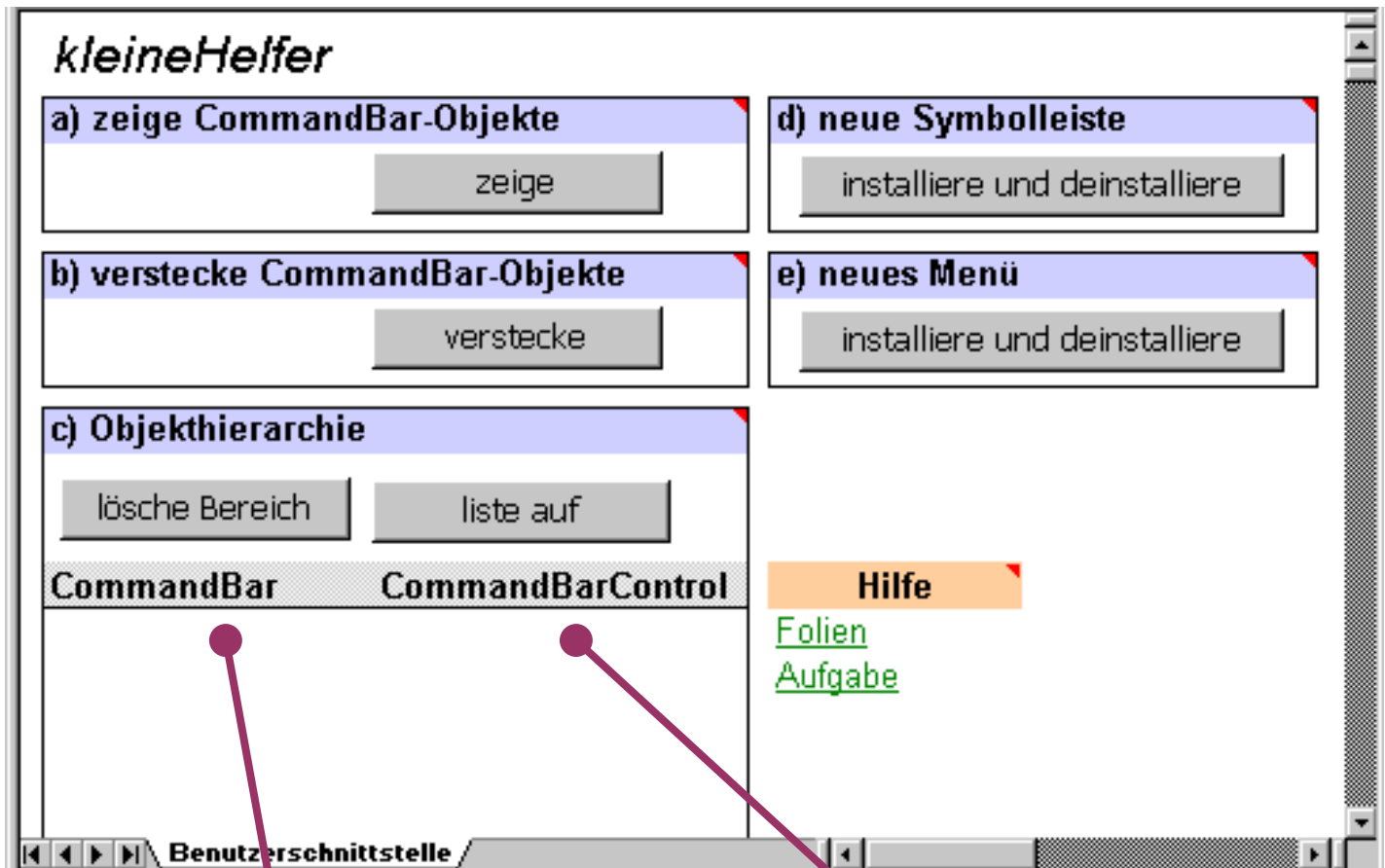
## CommandBars

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

## CommandBarControls

Eine Auflistung von **CommandBarControl**-Objekten, die die Befehlsleisten-Steuerelemente (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

---

## 3. Woche

---

### Vorlesung: Objektvariablen und Objektklassen

Objekt, Auflistung, Objektmodell (Repetition)

Objektvariablen

- Verweis

- Vereinbarung

- Zuweisung

  - Übergabe als Argument, Rückgabe als Funktionswert

- Vergleich

Objektklassen

- Objekte erstellen

- Beispiel Telefonverzeichnis

  - Idee

  - Abstraktion, Entwurf, Objektmodell

  - Implementation

    - Objektklasse und Objekt

    - Vordefinierte Objektklasse "Collection"

      - Neue Objekte erstellen

    - Benutzerdefinierte Objektklasse "cEintrag"

      - Objektklasse definieren

      - Neue Objekte erstellen

    - Projektübersicht

    - Sortieralgorithmus

### Übung: Telefonverzeichnis

# Objekt und Auflistung

## Objekt

Beispiele:

- Application
- Range

<u>Tabelle1 : Worksheet</u>		
Name, Visible = True, Range = <table><tr><td><u>:Range</u></td></tr><tr><td></td></tr></table>	<u>:Range</u>	
<u>:Range</u>		
Activate, Delete, SaveAs		

## Auflistung und Objekt

Beispiele:

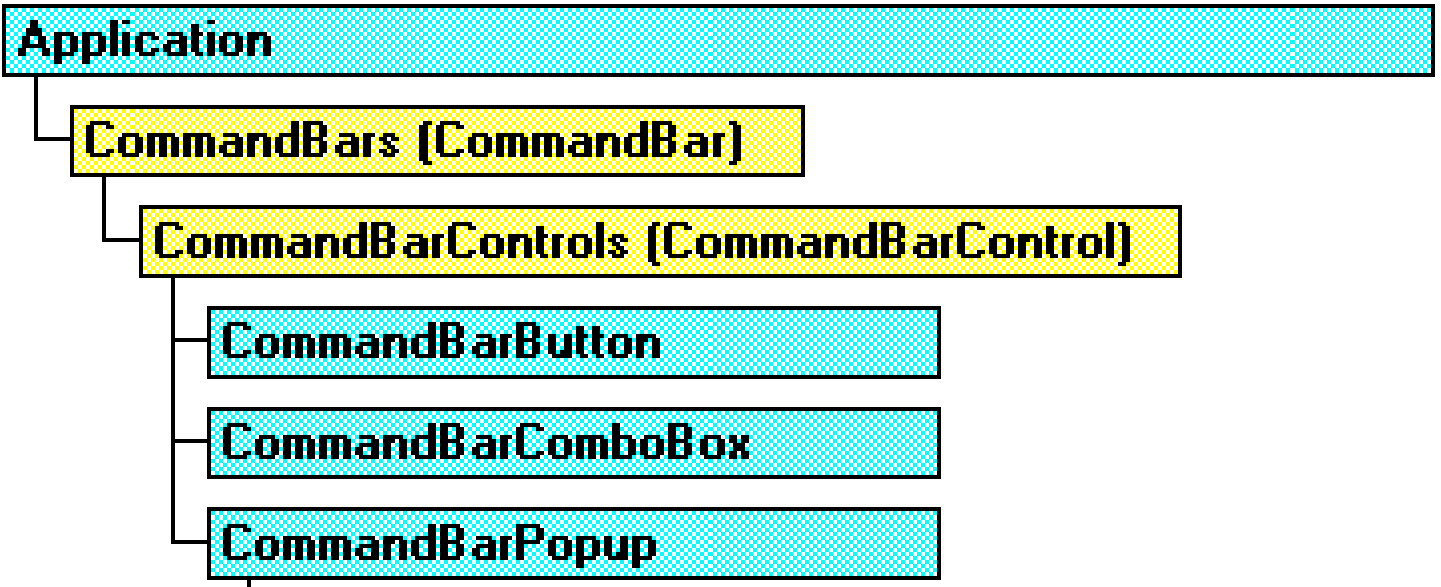
- Worksheets
- Workbooks
- CommandBars
- CommandBarControls

<u>:Worksheets</u>		
Item(1) = <table><tr><td><u>:Worksheet</u></td></tr><tr><td></td></tr></table>	<u>:Worksheet</u>	
<u>:Worksheet</u>		
Item(2) = <table><tr><td><u>:Worksheet</u></td></tr><tr><td></td></tr></table>	<u>:Worksheet</u>	
<u>:Worksheet</u>		
Count		
Add		



# Objekt und Objektmodell

## Objektmodell



## Zugriff auf einzelnes Objekt

⇒ Punktoperator

## Zugriff auf alle Objekte

⇒ For Each ... Next-Schleife

```
Dim BefehlsLeiste As CommandBar
Dim BefehlsFläche As CommandBarControl
For Each BefehlsLeiste In CommandBars
    ...
    For Each BefehlsFläche In befehlsLeiste.Controls
        ...
    Next BefehlsFläche
    ...
Next BefehlsLeiste
```

# Objektvariable

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

**Fangfrage:** Welche Variable enthält das Objekt  
"Zwischenresultate" am Ende des folgenden Programms?

```
Sub Test()  
    Dim Tab1 As Worksheet  
    Dim Tab2 As Worksheet  
  
    For Each Tab1 In Worksheets  
        If Tab1.Name = "Zwischenresultate" Then  
            Exit For  
        End If  
    Next Tab1  
  
    For Each Tab2 In Worksheets  
        If Tab2.Name = "Zwischenresultate" Then  
            Exit For  
        End If  
    Next Tab2  
  
End Sub
```

' (a)

' (b)

' (c)

# Objektvariablen und Verweise

Eine **Objektvariable** ist eine Variable, die einen Verweis (Zeiger) auf ein Objekt speichern kann.

Ein **Verweis** entspricht der Anfangsadresse eines Speicherbereiches, in dem sich ein Objekt befindet.

⇒ speichert *nicht* das Objekt selbst!

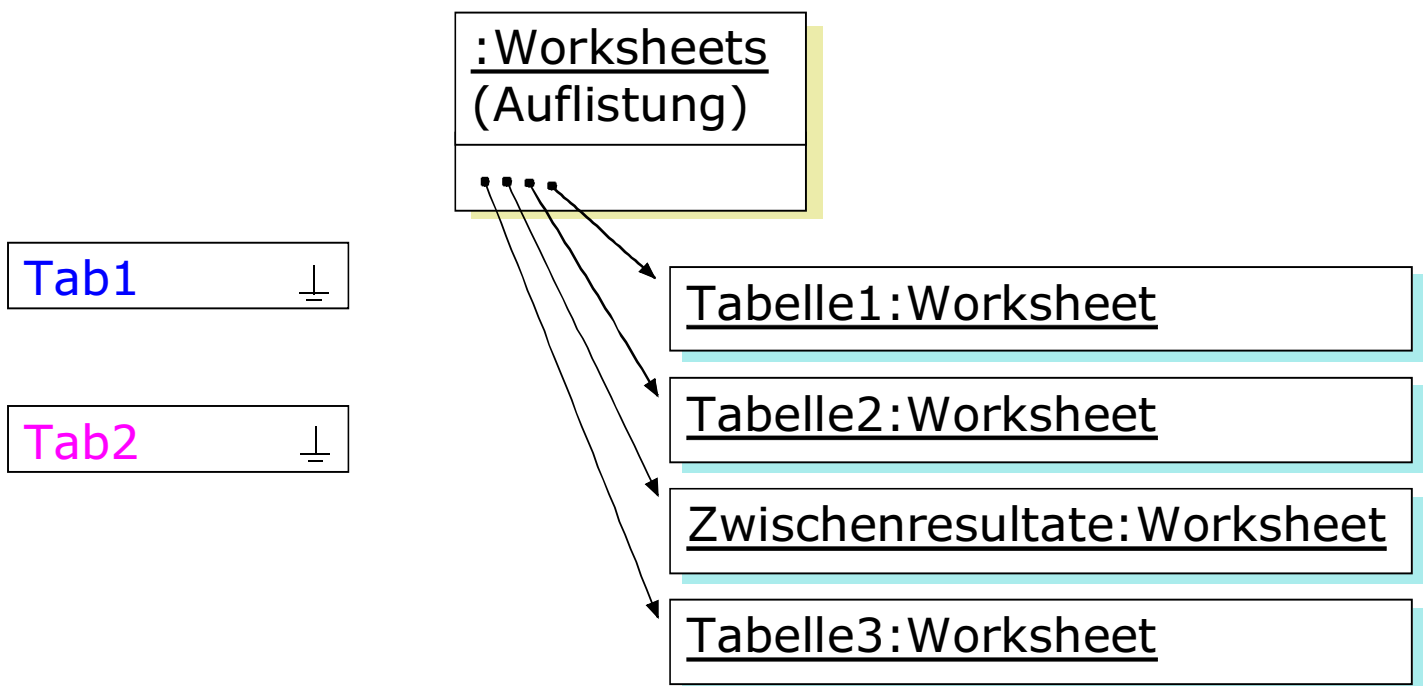
⇒ verweist nach der Vereinbarung auf **“Nothing”**

## Beispiel

Objekte, Objektvariable und Verweise im vergangenen Beispiel

Objektvariablen:

Objekte:



# Variablenvereinbarung

Objektvariable werden wie 'normale' Variable mit den Schlüsselwörtern **Dim**, **Private** oder **Public** vereinbart.

Zulässige Datentypen für Objektvariablen sind *spezifische* Objekttypen oder der *allgemeine* Objekttyp **Object**.

## Beispiele

<b>Dim</b> Blatt <b>As</b> Worksheet	<i>'spezifisch; Prozedurebene</i>
<b>Dim</b> Blatt <b>As</b> Object	<i>'allgemein; Prozedurebene</i>
<b>Public</b> Blatt <b>As</b> Worksheet	<i>'spezifisch; Projektebene</i>
<b>Private</b> Blatt <b>As</b> Object	<i>'allgemein; Modulebene</i>

# Variablenzuweisung

Die Zuweisung an eine Objektvariable muss immer mit dem Schlüsselwort **Set** eingeleitet werden.

## Beispiel

	A	B	
1			
2			

```
Dim Zelle1 As Range
```

```
Dim Zelle2 As Range
```

```
Set Zelle1 = Worksheets(1).Cells(1, 1)
```

```
Set Zelle2 = Worksheets(1).Cells(1, 2)
```

```
Zelle1.Value = "Text1"
```

```
Zelle2.Value = "Text2"
```

```
Set Zelle2 = Zelle1
```

```
Zelle1.Value = "Text3"
```

```
Zelle2.Value = "Text4"
```

### Standardeigenschaften:

```
Zelle2 = Zelle1
```

entspricht

```
Zelle2.Value = Zelle1.Value
```

# Argumente und Funktionswerte

## Funktionswerte (Zwischenresultate II.xls)

Wir lagern den Programmcode, der das Tabellenblatt "Zwischenresultate" sucht, in eine Prozedur aus.

```
Function sucheBlatt(TabName As String) As Worksheet
    Dim Blatt As Worksheet
    For Each Blatt In Worksheets
        If Blatt.Name = TabName Then
            Set sucheBlatt = Blatt
        End If
    Next Blatt
End Function
```

Prozeduraufruf

```
Dim ZRBlatt As Worksheet
Set ZRBlatt = sucheBlatt("Zwischenresultate")
```

## Argumente (vertausche.xls)

```
Sub vertauscheBlätter(Blatt1 As Worksheet, _
    Blatt2 As Worksheet)
    ...
```

**End Sub**

Prozeduraufruf

```
Dim Tabelle1 As Worksheet
Dim Tabelle2 As Worksheet
Set Tabelle1 = Worksheets("Tab5")
Set Tabelle2 = Worksheets("Tab2")
vertauscheBlätter Tabelle1, Tabelle2
```

ohne Set

# Vergleichsoperator

Objektvariablen können mit dem Operator **Is** auf Gleichheit geprüft werden (*nicht* mit dem Operator =).

Jede Objektvariable kann mit **Nothing** verglichen werden.

## Beispiel (Zwischenresultate II.xls)

**Makro:** Erstelle eine Arbeitsblatt "Zwischenresultate", **falls** es noch nicht existiert. **Sonst** lösche es.

```
Sub Tabellenblatt_Klick()  
Dim ZRBlatt As Worksheet  
    Set ZRBlatt = sucheBlatt("Zwischenresultate")  
    If ZRBlatt Is Nothing Then  
        Worksheets.Add.Name = "Zwischenresultate"  
    Else  
        ZRBlatt.Delete  
    End If  
End Sub
```

Achtung bei Standardeigenschaften!

```
If Zelle1 = Zelle2 Then  
    entspricht nicht  
If Zelle1 Is Zelle2 Then
```

---

# Objekte erstellen

---

## vier Möglichkeiten, Objekte zu erstellen

- 1) Viele Objekte werden beim Start der Anwendung (z.B. Excel) erstellt
  - ✓ Application-Object
  - ✓ Workbooks-Auflistung (mit einem Workbook-Objekt)
  - ✓ CommandBars-Auflistung (mit CommandBar-Objekten)usw.
  
- 2) Die **Add**-Methode einer Auflistung erzeugt neue Objekte und fügt sie zur Auflistung hinzu.
  - ✓ Worksheets.**Add**.Name = "Zwischenresultate"usw.
  
- 3) Mit dem **New**-Schlüsselwort können benutzerdefinierte Objekte erstellt werden.  
⇒ **Set** VektorA = **New** clsVektor
  
- 4) Die **CreateObject**-Funktion erzeugt (ActiveX-) Objekte fremder Anwendungen (z.B. Access) und gibt Verweise darauf zurück.  
**Set** AccessApp = **CreateObject**("Access.Application")



# Telefonverzeichnis.xls

## Aufgabe

- Eingabe von Namen und Telefonnummern
- Verwalten der Einträge in einem Verzeichnis
- Erstellen einer alphabetischen Liste aller Einträge

## Benutzerschnittstelle

### Eingabebereiche

- Range("name")
- Range("telefon")

name	telefon
Heidi	43 43 03

### Verarbeitung

- hinzufügen\_Klick()
- suchen\_Klick()
- auflisten\_Klick()

anzahl		4
liste_name	liste_telefon	
Gerald	74 12 91	
Heidi	43 43 03	
Peter	58 92 29	
Sabine	78 98 12	

### Ausgabebereiche

- Range("anzahl")
- Range("telefon")
- Range("liste\_name")
- Range("liste\_telefon")

# Abstraktion

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

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

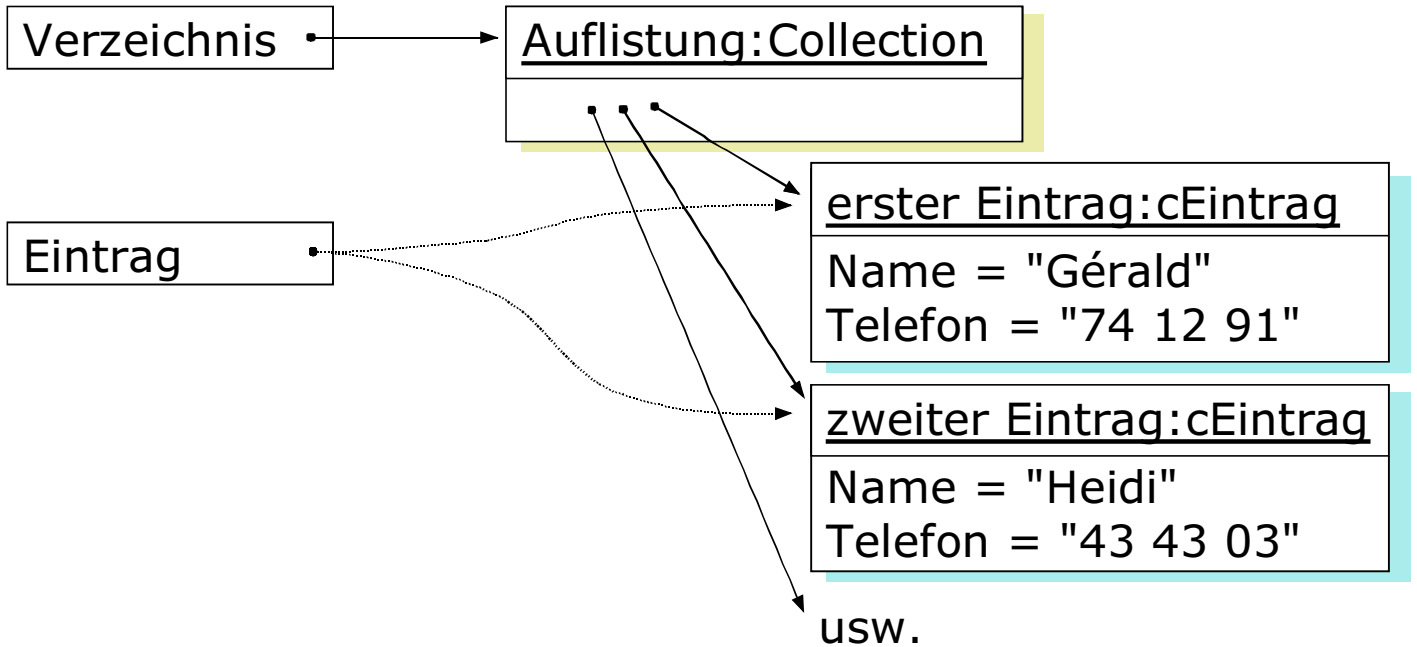
Als **Abstraktion** bezeichnet man die selektive Auswahl bestimmter Aspekte eines Gegenstandes oder eines Problems, mit dem Ziel, wichtige Aspekte zu isolieren und unwichtige zu ignorieren.

⇒ Karteikasten mit Karten

# Objektmodell

Objektvariablen:

Objekte:



## Wir brauchen ...

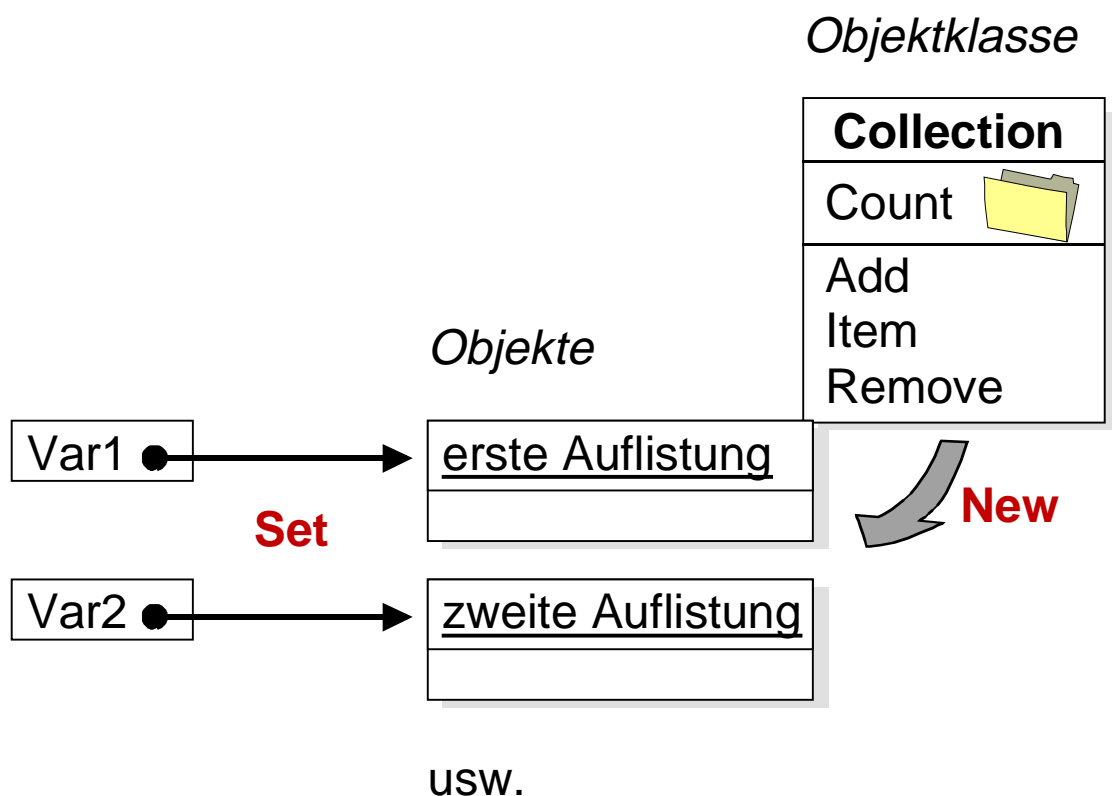
- ⇒ mehrere 'Eintrag'-**Objekte**, welche die Karten des Karteikartensystems repräsentieren
- ⇒ ein 'Auflistung'-**Objekt**, welches den Karteikasten repräsentiert
- ⇒ eine **Objektvariable** 'Eintrag', um auf die 'Eintrag'-**Objekte** zugreifen zu können.
- ⇒ eine **Objektvariable** 'Verzeichnis', um auf das 'Auflistung'-**Objekt** zugreifen zu können
- ⇒ einen **Algorithmus**, der die Einträge im Verzeichnis alphabetisch **sortiert**.

# Objektklasse und Objekt

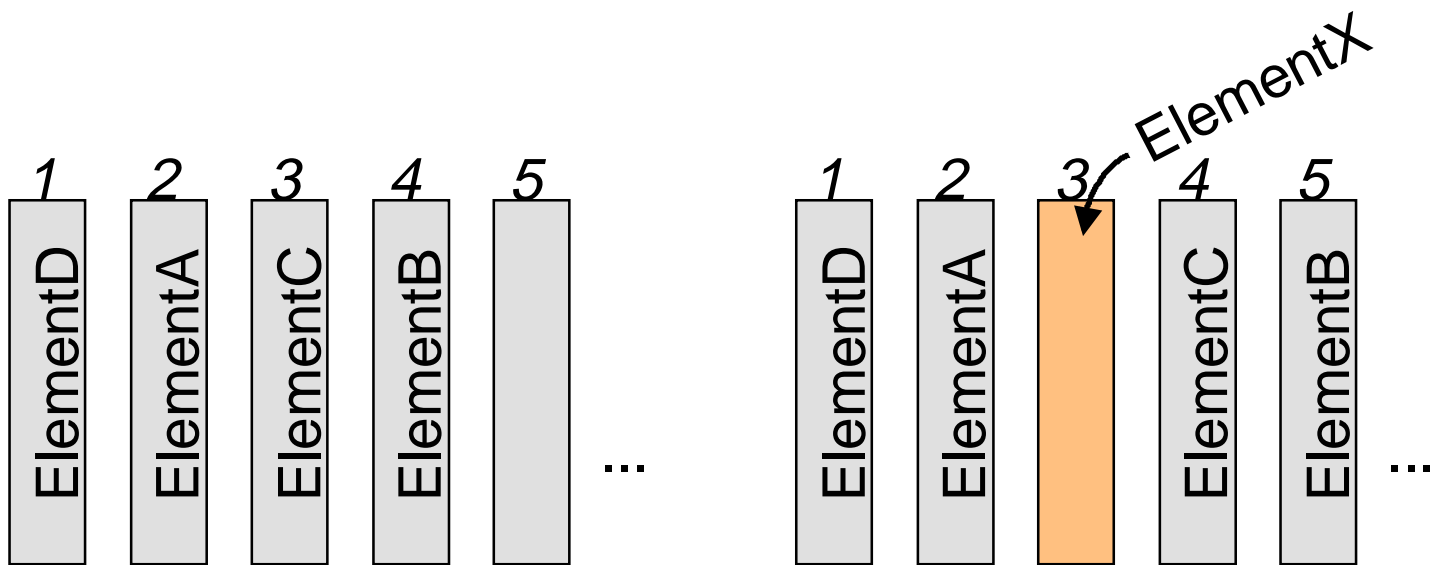
Eine **Objektklasse** ist eine **Beschreibung** einer Gruppe von Objekten mit gleichen Eigenschaften und gleichem Verhalten. Der *Zustand* der Objekte wird durch eine Menge von *Eigenschaften* (Daten), das *Verhalten* durch eine Menge von *Methoden* (Prozeduren) beschrieben.

## Beispiel

Visual Basic stellt die Objektklasse **Collection** für Auflistungsobjekte zur Verfügung



# Objektklasse “Collection”



Ein **Collection**-Objekt ist

- eine *geordnete* Folge  
(→Vorgänger und Nachfolger)
- von *Elementen*, auf die  
(→beliebige Datentypen, auch gemischt)
- als Einheit Bezug genommen werden kann.

## Eigenschaften

- **Objekt.Count** (schreibgeschützt)

## Methoden

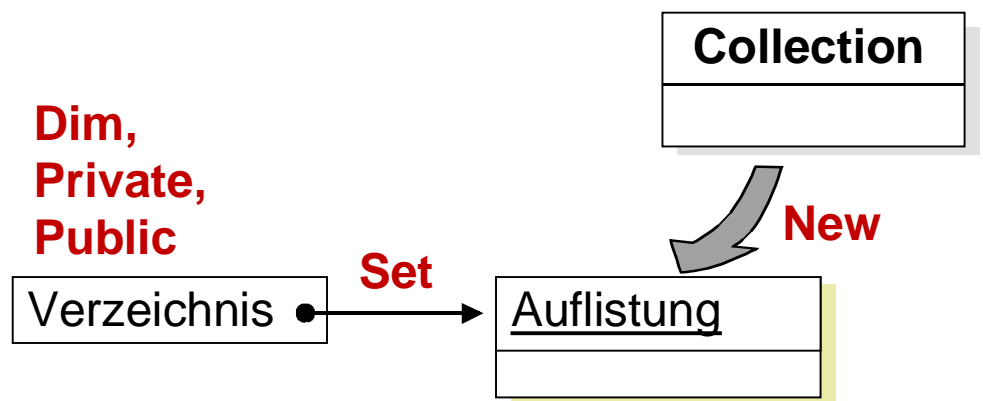
- **Objekt.Add** (item, key, before, after) (benannte Argumente)
- **Objekt.Item**(Index)
- **Objekt.Remove** Index

# Neue Objekte erstellen

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

Die **New**-Anweisung erzeugt eine Instanz einer Objektklasse

## drei Schritte



- 1) Objektvariable vereinbaren
- 2) Objekt erstellen und der Variablen zuweisen
- 3) über Variable auf Methoden und Eigenschaften zugreifen

## Beispiel Telefonverzeichnis.xls

- 1) Im Vereinbarungsabschnitt

**Private** Verzeichnis **As** Collection

- 2) Innerhalb einer Prozedur

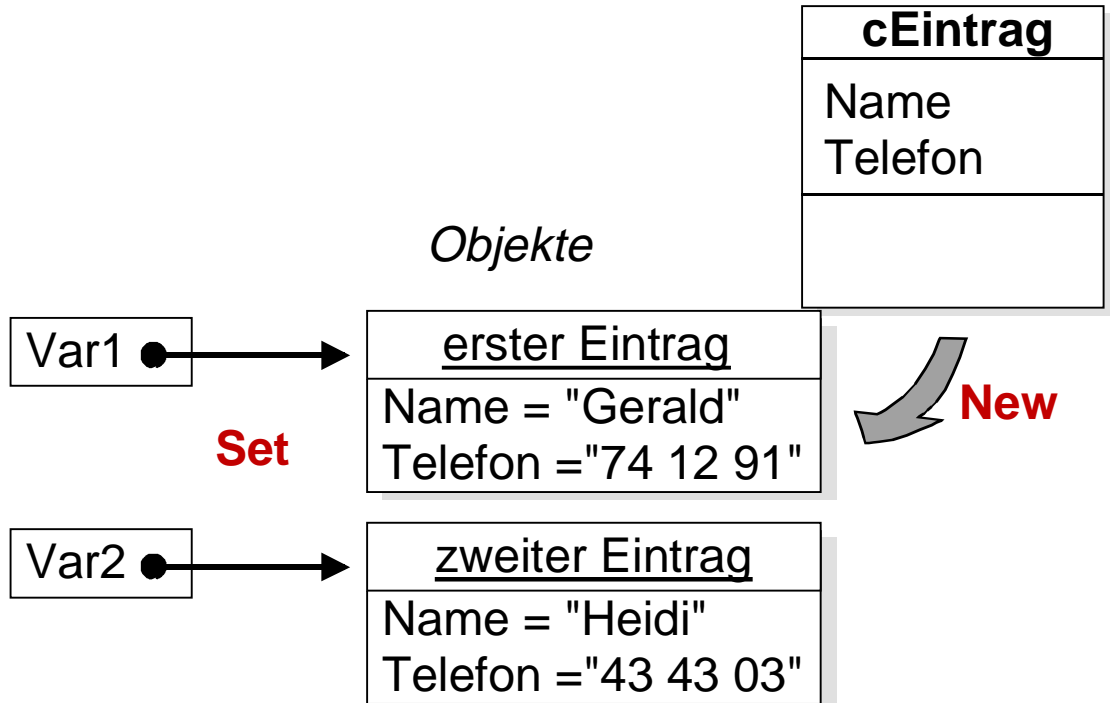
**Set** Verzeichnis = **New** Collection

- 3) Innerhalb einer Prozedur

z.B.: Range ("anzahl") = Verzeichnis.Count

# Benutzerdefinierte Objektklasse

## Beispiel: Objektklasse "cEintrag"



Ein **Klassenmodul** ist ein Modul, das die Definition einer Objektklasse enthält, einschliesslich der Definitionen seiner Eigenschaft und Methoden.

# Benutzerdefinierte Objektklasse “cEintrag”

- 1) neues Klassenmodul erstellen
- 2) Klassenmodul benennen



- 3) Eigenschaften und Methoden implementieren

**Eigenschaften** sind im wesentlichen öffentliche Variablen,  
**Methoden** sind öffentliche Prozeduren.

## Option Explicit

*' --- EIGENSCHAFTEN*

**Public** Name **As String**

**Public** Telefon **As String**

*' --- METHODEN*



---

# Neue Objekte erstellen

---

## wieder drei Schritte

1) Objektvariable deklarieren

```
Dim neuerEintrag As cEintrag
```

2) Objekt erstellen und der Variablen zuweisen

```
Set neuerEintrag = New cEintrag
```

3) über Variable auf (Methoden und) Eigenschaften zugreifen

```
neuerEintrag.Name = Range("name")
```

```
neuerEintrag.Telefon = Range("telefon")
```

neues Objekt in Auflistung verwalten

```
Verzeichnis.Add neuerEintrag
```

# Implementation des Telefonverzeichnisses

## Projekt

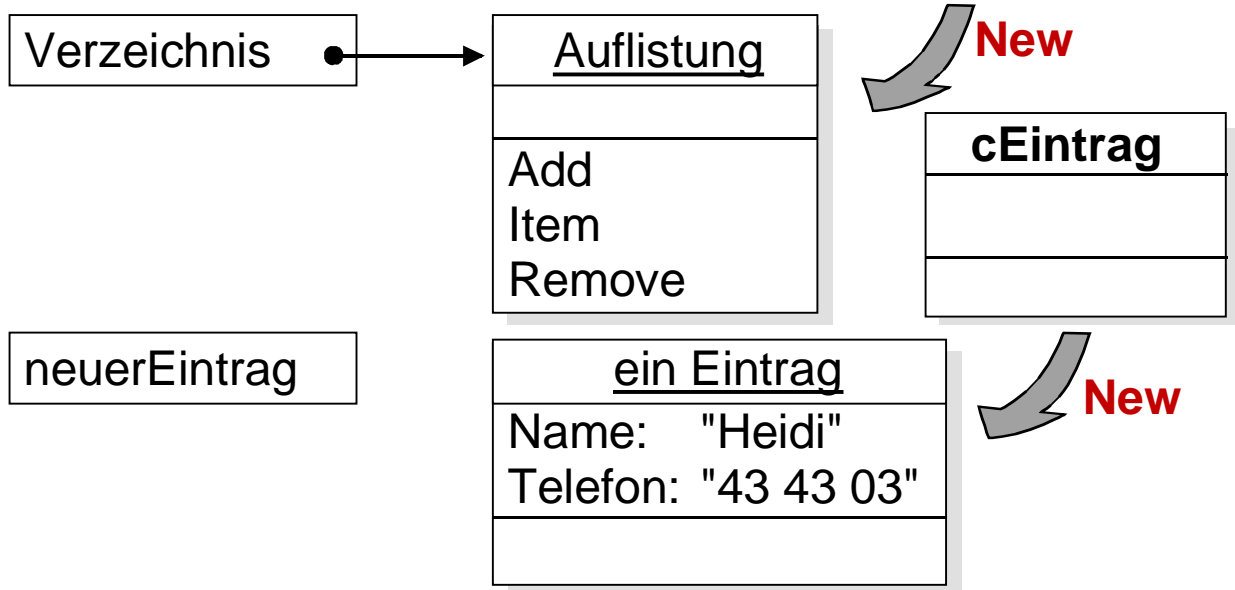


## Ablauf

*Objektvariablen:*

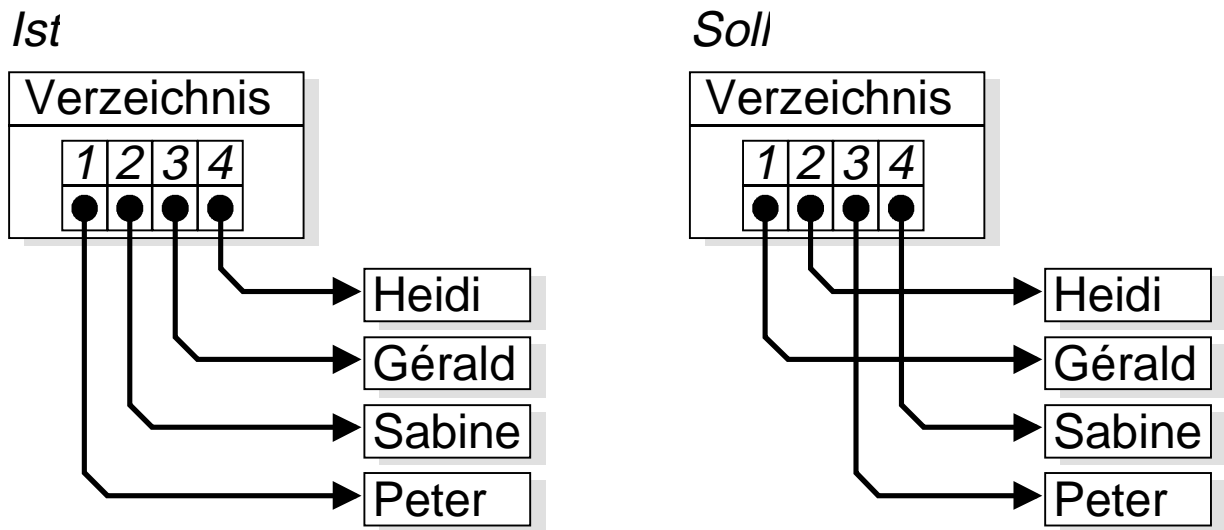
*Objekte:*

*Objektklassen:*



# Sortieralgorithmus

## Beispiel



## Sortieralgorithmus

### sortiere durch Auswählen

erstelle neues Verzeichnis (ohne Einträge)

**SOLANGE** noch Einträge im alten Verzeichnis

**suche kleinsten Eintrag** im alten Verzeichnis

füge Eintrag am Ende des neuen Verzeichnis ein

entferne Eintrag aus altem Verzeichnis

ersetze altes Verzeichnis durch neues Verzeichnis.

**suche kleinsten Eintrag** im Verzeichnis

setze kleinster Eintrag = erster Eintrag

**FÜR ALLE** Einträge im Verzeichnis

**FALLS** aktueller Eintrag < kleinster Eintrag

setze kleinster Eintrag = aktueller Eintrag.

---

## 4. Woche

---

### Vorlesung: Benutzerdefinierte Objektklassen

Repetition: Objektvariable, Verweis, Objekte erstellen

Objekte löschen

Benutzerdefinierte Objektklassen

- Klassenmodul

  - Modul

  - Eigenschaften beschreiben

    - Property Get

    - Property Let und Property Set

  - Methoden beschreiben

  - Konstruktor

  - Destruktor

- Meldungen und Ereignisse

- Klassen und Instanzen

- Identität

Objektorientierte Programmiersprachen

- Merkmale

  - Kapselung

  - Abstraktion

  - Polymorphismus

  - Vererbung

### Übung: Geburtstage

# Zusammenfassung

## Objektvariablen und Verweise

Eine **Objektvariable** ist eine Variable, die einen Verweis (Zeiger) auf ein Objekt speichern kann.

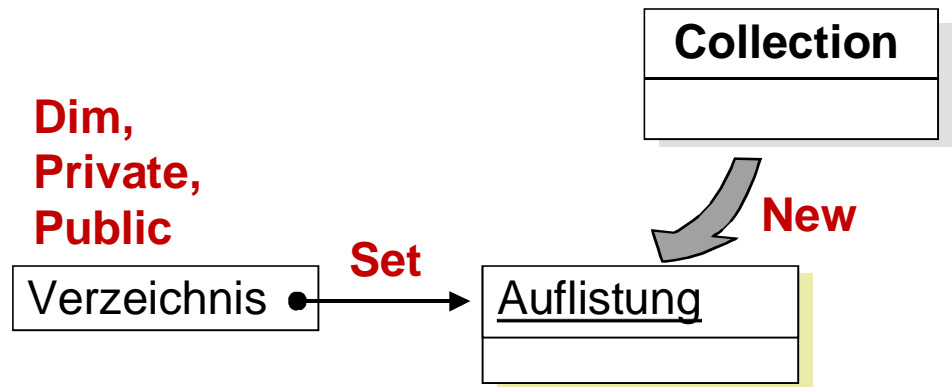
Ein **Verweis** entspricht der Anfangsadresse eines Speicherbereiches, in dem sich ein Objekt befindet.

- ⇒ speichert *nicht* das Objekt selbst!
- ⇒ Vereinbarung mit **spezifischem** oder **allgemeinem** Objekttyp
- ⇒ verweist nach der Vereinbarung auf '**Nothing**'
- ⇒ Zuweisungsanweisungen mit **Set** einleiten
- ⇒ Vergleiche mit **Is**-Operator

# Zusammenfassung

## Objekte erstellen

### drei Schritte



- 1) Objektvariable vereinbaren
- 2) Objekt erstellen und der Variablen zuweisen
- 3) über Variable auf Methoden und Eigenschaften zugreifen

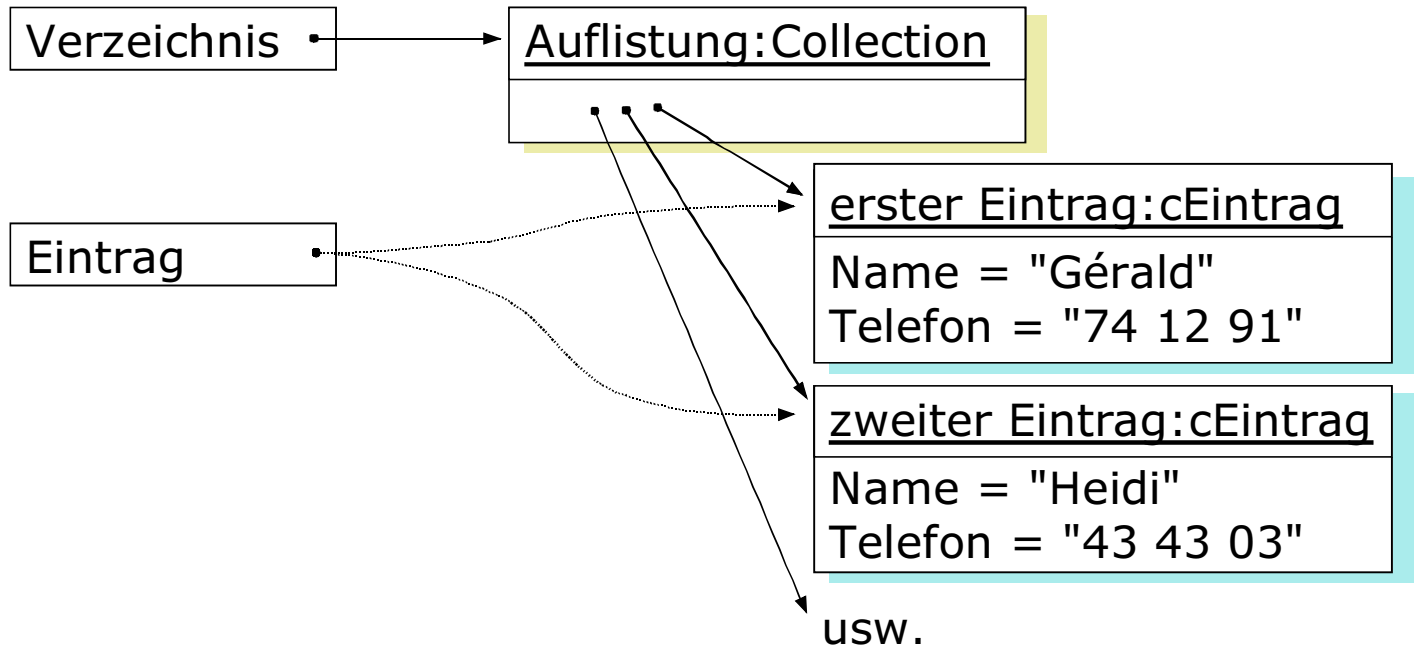
## Beispiel

```
Dim neuerEintrag As cEintrag
Set neuerEintrag = New cEintrag
neuerEintrag.Name = Range("name")
neuerEintrag.Telefon = Range("telefon")
```

# Objektmodell des Telefonverzeichnisses

Objektvariablen:

Objekte:



## Wir brauchen ...

- ein 'Auflistung'-**Objekt**, welches den Karteikasten repräsentiert  
⇒ *Collection-Klasse*
- mehrere 'Eintrag'-**Objekte**, welche die Karten des Karteikartensystems repräsentieren  
⇒ *benutzerdefinierte Klasse, Klassenmodul*
- eine **Objektvariable** 'Verzeichnis', um auf das 'Auflistung'-**Objekt** zugreifen zu können
- eine **Objektvariable** 'Eintrag', um auf die 'Eintrag'-**Objekte** zugreifen zu können.
- einen **Algorithmus**, der die Einträge im Verzeichnis alphabetisch **sortiert**.  
⇒ *zweite Instanz des Verzeichnisses*

# Arbeiten mit Objektverweisen

## Erstes Beispiel

Auf welche Objekte verweisen die Variablen im folgenden Programm, jeweils nach Ausführung der nummerierten Zeilen?

```
Private Eintrag1 As cEintrag
Private Eintrag2 As cEintrag
Sub Test()
    Dim Temp As cEintrag           '1
    Set Eintrag1 = New cEintrag    '2
    Set Eintrag2 = New cEintrag    '3
    Set Temp = Eintrag1           '4
    Set Eintrag1 = Eintrag2       '5
    Set Eintrag2 = Temp           '6
End Sub                           '7
```

*Objektvariablen:*

*Objekte:*

Eintrag1

Eintrag2

Temp



# Arbeiten mit Objektverweisen

## Zweites Beispiel

Auf welche Objekte verweisen die Variablen im folgenden Programm, nach Ausführung der Prozedur Test?

```
Sub Test()  
    Dim Temp As cEintrag  
    Set Temp = New cEintrag  
End Sub
```

Objektvariablen:

Objekte:

Temp

## Konsequenz: Löschen von Objekten

Ein Objekt wird **automatisch** gelöscht, wenn keine Variable mehr darauf verweist.

⇒ Objekte können mit Hilfe des Schlüsselworts **Nothing** explizit gelöscht werden

### Beispiel

```
Private Temp As cEintrag  
Sub Test()  
    Set Temp = New cEintrag  
    ...  
    Set Temp = Nothing  
End Sub
```

# Objektklassen in Telefonverzeichnis.xls

## zur Erinnerung

Ein **Objekt** ist eine Einheit aus **Daten** (**Eigenschaften** + **Werte**) und **Operationen** (**Methoden**).

Eine **Objektklasse** ist eine Beschreibung von **Eigenschaften** und **Methoden** zur Erzeugung von Objekten des gleichen Typs.

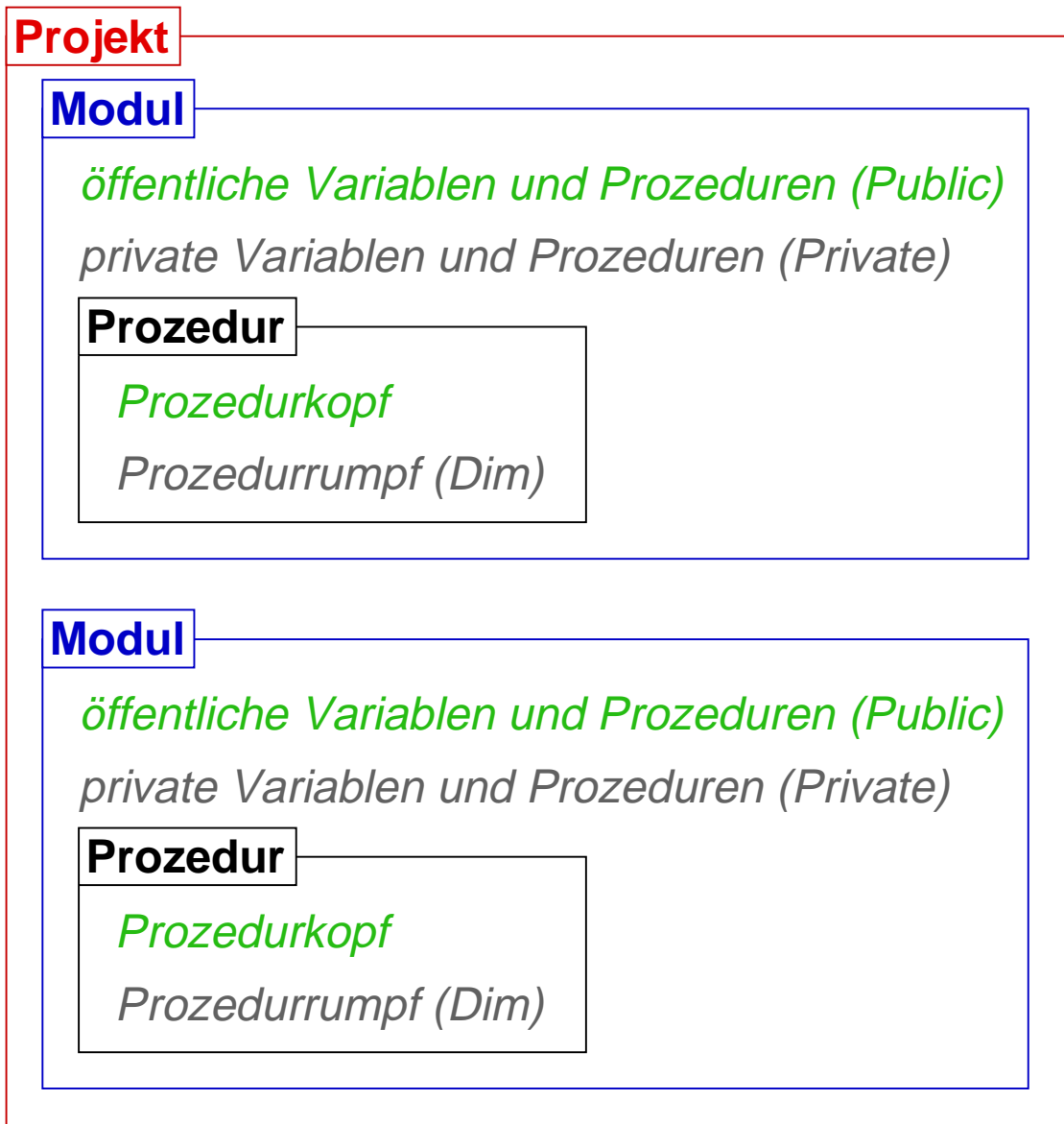
## Vergleich zwischen Collection und cEintrag

	<i>Collection</i>	<i>cEintrag</i>
mehrere Instanzen	✓	✓
Daten (Eigenschaften):		
öffentlich	✗	✓ z.B. Name
öffentlich, schreibgeschützt	✓ z.B. count	✗
öffentlich, lesegeschützt	✗	✗
privat	?	✗
Operationen (Methoden):		
öffentlich, Sub-	✓ z.B. Add	✗
öffentlich, Function-	✓ z.B. Item	✗
privat	?	✗

# Klassenmodule sind Module

**Module** trennen eine unsichtbare und unantastbare **Implementation** von einer sichtbaren und zugänglichen **Schnittstelle**

- ⇒ VBA-Onlinehilfe beschreibt **Schnittstelle** der Collection-Klasse
- ⇒ Implementation wird geheim gehalten (engl. information hiding)



# Eigenschaften beschreiben

## durch öffentliche Variablen

z.B.: **Beschreibung** im Klassenmodul  
**Public** Name **As String**

**Verwendung** im Benutzermodul

*Objekt.Name = Ausdruck*

*Variable = Objekt.Name*

- 😊 einfach
- 😞 jederzeit abruf- und änderbar
- 😞 keine Validitätsprüfung bei Zuweisung

## mit Property-Prozeduren

z.B.: **Beschreibung** im Klassenmodul  
(→ siehe nächste Folien)

**Verwendung** im Benutzermodul

*Objekt.Name = Ausdruck*

*Variable = Objekt.Name*

- 😞 aufwändiger
- 😊 individuelle Schreib- und Leseberechtigungen
- 😊 Validitätsprüfung und Korrektur bei Zuweisung

# Property Prozeduren

**Property Prozeduren** definieren Eigenschaften mit individueller Schreib- und Leseberechtigung

## 1. öffentliche Property-Prozeduren

→ **Schnittstelle**

<i>Property Prozedur</i>	<i>Zugriff</i>	<i>Variablenart</i>
Property Get	lesen	'normale' und Objektvariablen
Property Let	schreiben	'normale' Variablen
Property Set	schreiben	Objektvariablen

## 2. private Eigenschaftsvariablen

→ **Implementation**

z.B.: **Beschreibung** im Klassenmodul

'normale' Variablen

**Private** strName **As String**

**Private** strTelefon **As String**

Objektvariablen

**Private** Mutter **As** cEintrag

**Private** Vater **As** cEintrag

---

# Property Get

---

## 2a Lesezugriff für 'normale' Variablen

```
Public Property Get Eigenschaft() As Typ  
    [Eigenschaft = Eigenschaftsvariable]  
    [andere Anweisungen]  
End Property
```

z.B.: **Beschreibung** im Klassenmodul

```
Public Property Get Name As String  
    Name = strName  
End Property
```

**Verwendung** im Benutzermodul

```
Variable = Objekt.Name
```

## 2b Lesezugriff für Objektvariablen

```
Public Property Get Eigenschaft() As Objekttyp  
    [Set Eigenschaft = Eigenschaftsvariable]  
    [andere Anweisungen]  
End Property
```

**Verwendung** im Benutzermodul

```
Set Objektvariable = Objekt.Eigenschaft
```

⇒ **Vereinbarung** wie Function-Prozedur

⇒ **Benutzung** wie Variable bzw. Objektvariable

# Property Let und Property Set

## 2a Schreibzugriff für 'normale' Variablen

```
Public Property Let Eigenschaft(Arg As Typ)
    [Eigenschaftsvariable = Arg]
    [andere Anweisungen]
End Property
```

z.B.: **Beschreibung** im Klassenmodul

```
Public Property Let Name(neuerName As String)
    strName = Trim(neuerName)
End Property
```

**Verwendung** im Benutzermodul

```
Objekt.Name = Ausdruck
```

## 2b Schreibzugriff für Objektvariablen

```
Public Property Set Eigenschaft(Arg As Objekttyp)
    [Set Eigenschaftsvariable = Arg]
    [andere Anweisungen]
End Property
```

**Verwendung** im Benutzermodul

```
Set Objekt.Eigenschaft = Ausdruck
```

⇒ **Vereinbarung** wie Sub-Prozedur

⇒ **Benutzung** wie Variable bzw. Objektvariable

# Bemerkungen

## zu Eigenschaften

- ⇒ Eine Objektklasse beschreibt nur die **Eigenschaften**.
- ⇒ In den Instanzen einer Klasse hat jede Eigenschaft einen **Wert**.

**Zustand** := alle **Eigenschaften** mit ihren  
aktuellen **Werten**

- ⇒ Der Zustand eines Objektes ändert sich, sobald sich der Wert einer Eigenschaft ändert.

## zu Property-Prozeduren

- ⇒ Eine Eigenschaft besitzt oft eine Get- *und* eine Let- (oder Set-) Property mit *demselben* Namen.
- ⇒ Eine Eigenschaft ohne Get-Property ist *lesegeschützt*.
- ⇒ Eine Eigenschaft ohne Let-Property ist *schreibgeschützt*.
- ⇒ Eigenschaft und Eigenschaftsvariablen können *nicht* denselben Namen haben.



# Methoden beschreiben

## durch öffentliche Prozeduren

z.B.: **Beschreibung** im Klassenmodul

```
Public Sub wähleNummer()  
    ...  
End Sub
```

**Verwendung** im Benutzermodul

```
Verzeichnis.Item(2).wähleNummer
```

z.B.: **Beschreibung** im Klassenmodul

```
Public Function wähleNummer() As Boolean  
    ...  
    'WENN nach 30sec Hörer abgehoben wurde  
    '    wähleNummer = True  
    'SONST  
    '    wähleNummer = False  
End Function
```

**Verwendung** im Benutzermodul

```
If Verzeichnis.Item(2).wähleNummer Then  
    'begrüsse Item(2) und  
    'telefoniere mit ihr/ihm
```

⇒ Methoden beschreiben das *Verhalten* von Objekten

- führen Aktionen aus
- ändern den Zustand des Objektes

# Konstruktor

Ein **Konstruktor** ist eine Methode, die ein Objekt bei der Erzeugung initialisiert.

## Beschreibung im Klassenmodul

```
Private Sub Class_Initialize()  
    ...  
End Sub
```

- ⇒ wird automatisch unmittelbar nach der Erzeugung eines Objektes (mit New) ausgeführt
- ⇒ Prozedurkopf ist vordefiniert (Name, Gültigkeitsbereich und Argumente)
- ⇒ kann nicht explizit aufgerufen werden
- ⇒ dient der Initialisierung

## Beispiel

```
Private Sub Class_Initialize()  
    strName = ""  
    strTelefon = "hat kein Telefon"  
End Sub
```

# Destruktor

Ein **Destruktor** ist eine Methode, die den Zustand eines Objektes freigibt.

## Beschreibung im Klassenmodul

```
Private Sub Class_Terminate()  
    ...  
End Sub
```

- ⇒ wird automatisch unmittelbar vor der Zerstörung eines Objektes ausgeführt
- ⇒ Prozedurkopf ist vordefiniert (Name, Gültigkeitsbereich und Argumente)
- ⇒ kann nicht explizit aufgerufen werden
- ⇒ dient
  - zur Sicherung der Daten und
  - zum Informieren anderer Objekte

## Beispiel

```
Private Sub Class_Terminate()  
    MsgBox "Eintrag " & Name & " wird gelöscht! " & _  
        "Möchten Sie sich die Telefonnummer " & _  
        "vorher notieren?"  
    ...  
End Sub
```

# Arbeiten mit Objektverweisen

## Drittes Beispiel

Klassenmodul: cEintrag

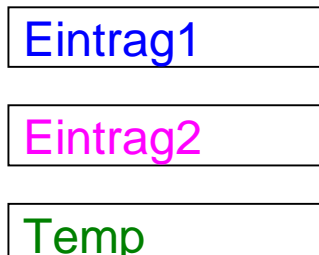
```
Private Sub Class_Initialize()  
    MsgBox "Objekt wurde erstellt"  
End Sub  
  
Private Sub Class_Terminate()  
    MsgBox "Objekt wird gelöscht"  
End Sub
```

Codemodul: Test

```
Public Eintrag1 As cEintrag  
Private Eintrag2 As cEintrag  
  
Sub Test() '1  
    Dim Temp As cEintrag '2  
    Set Temp = New cEintrag '3  
    Set Eintrag1 = New cEintrag '4  
    Set Eintrag2 = Temp '5  
    Set Temp = Eintrag1 '6  
    Set Eintrag1 = Nothing '7  
End Sub
```

Objektvariablen:

Objekte:



# Methoden ausführen

## Punktoperator

Beispiel:

```
Verzeichnis.Item(2).wähleNummer
```

- ☹ Es muss bekannt sein, welche Methoden ein Objekt implementiert.

## Meldungen

Eine **Meldung** (engl. message) informiert über eine Veränderung in einem Objekt. Das Eintreffen einer Meldung löst beim Empfänger ein **Ereignis** aus.

Eine **Ereignisprozedur** ist eine Methode, die eine Reaktion auf ein Ereignis implementiert.

- ⇒ Beispiel: Click-Ereignis
- ⇒ Konstruktor und Destruktor sind Ereignisprozeduren
- ☺ Ereignisprozeduren können, müssen aber nicht implementiert werden.

# Klassenmodul und Instanzen

Von Klassenmodulen können Instanzen erstellt werden, von Codemodulen nicht.

## Was passiert beim Erstellen von Instanzen?

- ⇒ Jede Instanz braucht eigene Instanzvariablen.
- ⇒ Jede Instanz benutzt die Methoden der Klasse.

### Beispiel:

Klassenmodul: cEintrag

```
Public Name As String
```

```
Public Sub zeige()
```

```
    MsgBox Name
```

```
End Sub
```

Codemodul: Test

```
Private Sub Test()
```

```
    Dim Objekt1 As cEintrag
```

```
    Dim Objekt2 As cEintrag
```

```
    Set Objekt1 = New cEintrag
```

```
    Objekt1.Name = "anna"
```

```
    Set Objekt2 = New cEintrag
```

```
    Objekt2.Name = "beat"
```

```
    Objekt1.zeige
```

```
    Objekt2.zeige
```

```
End Sub
```

---

# 5. Woche

---

## Vorlesung: Verkettete Strukturen

Repetition: benutzerdefinierte Klassen, Instanzen

Identität von Instanzen

Objektorientierte Programmiersprachen

- Kapselung

- Abstraktion

- Polymorphismus

- Vererbung

Datenverwaltung

- Datenfeld

- verkettete Datenstrukturen

  - verkettete Liste

  - Baumstrukturen

    - Binärbaum (sortiert)

      - einfügen

      - suchen

      - der "ideale Binärbaum"

        - Höhe, Ausbalanciertheit

        - rekursive Datenstruktur

        - rekursive Algorithmen

## Übung: keine Übung (Auffahrt)

# Objekte beschreiben

---

## Abstraktion

- von realen Gegenständen oder Konzepten
- wichtiges isolieren
- unwichtiges ignorieren
- **Bsp.:** Karteikasten mit Karteikarten

## benutzerdefinierte Klasse: Klassenmodul

- **Eigenschaften** für Beschreibung des Zustands
  - ⇒ öffentliche Variablen im Klassenmodul
  - ⇒ Property Get-/Let-Prozeduren
- **Methoden** für Beschreibung des Verhaltens
  - ⇒ öffentliche Prozeduren im Klassenmodul
  - ⇒ Ereignisprozeduren
  - ⇒ Konstruktor/Destruktor



# Instanzen erstellen, Objektmodelle aufbauen

## Instanzen benutzerdefinierter Klassen

- 1) Objektvariable vereinbaren
- 2) Instanz erstellen und der Variablen zuweisen
- 3) Methoden und Eigenschaften benutzen

```
Bsp.: Dim neuerEintrag As cEintrag
      Set neuerEintrag = New cEintrag
      neuerEintrag.Name = Range("name")
```

## Objektmodell

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

- **statisch**
  - ⇒ einzelne Objekte als Eigenschaften
  - ⇒ **Bsp.:** *“Ein Velo hat zwei Räder”*
- **dynamisch**
  - ⇒ Collection als Eigenschaften
  - ⇒ **Bsp.:** *“Eine Firma hat MitarbeiterInnen”*

# Identität

Jedes Objekt besitzt eine Eigenschaft **Me**, die einen Verweis auf das Objekt selbst speichert.

- ⇒ Die Eigenschaft **Me** ist schreibgeschützt.
- ⇒ Der Inhalt von **Me** ist für jedes Objekt eindeutig und einzigartig.
- ⇒ Auch zwei *gleiche* Objekte unterscheiden sich im Inhalt der Variablen **Me**.
- ⇒ Man kann den Inhalt der Variablen **Me** nicht anschauen.
- ⇒ Man bezeichnet **Me** als Identität.

Die **Identität** ist eine Eigenschaft, die jedes Objekt besitzt, und die es von allen anderen unterscheidet.  
Zwei Objekte heissen **gleich**, wenn sie exakte Kopien voneinander sind.

Zwei Objekte heissen **identisch**, wenn sie dieselbe Identität haben.

- ⇒ Objektvariablen speichern Identitäten.
- ⇒ If Objekt1 **=** Objekt2 Then  
vergleicht auf Gleichheit.
- ⇒ If Objekt1 **Is** Objekt2 Then  
vergleicht die Identitäten.

# Objektorientierte Programmiersprachen

---

## Vier wichtige Merkmale

1. Kapselung (engl. encapsulation)
2. Abstraktion (engl. abstraction)
3. Polymorphismus (engl. polymorphism)
4. Vererbung (engl. inheritance)

⇒ Was bedeuten diese Begriffe?

⇒ Unterstützt auch Visual Basic diese Konzepte?

# Kapselung

**Kapselung** := Prinzip der Geheimhaltung des '*Wie*' (Implementation) und der Veröffentlichung des '*Was*' (Schnittstelle)

## in Visual Basic

⇒ *Gültigkeitsbereich* von Variablen und Prozeduren

⇒ *Property*-Prozeduren

## Beispiel

auf Prozedurebene

```
Function kleinstenEintrag() As Integer
    Dim i As Integer
    ...
End Function
```

auf Modulebene

```
Private strName As String
Public Property Get Name As String
    Name = strName
End Property
Public Property Let Name(neuerName As String)
    strName = Trim(neuerName)
End Property
```

# Abstraktion

**Abstraktion** := selektive Auswahl bestimmter Aspekte eines Gegenstands oder eines Problems

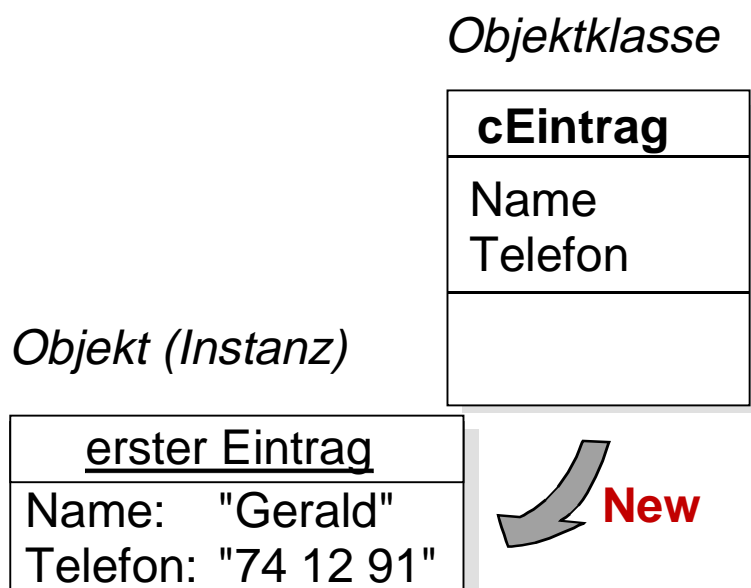
- ⇒ wichtige von unwichtigen Aspekten trennen
- ⇒ beschreiben von Eigenschaften und Methoden
- ⇒ modellieren der Wirklichkeit
- ⇒ kreieren abstrakter Datentypen

## in Visual Basic

- ⇒ Objekte in *Klassenmodulen* beschreiben
- ⇒ *Instanzen* mit aufgrund der Beschreibung erstellen

## Beispiel

Klasse cEintrag



# Polymorphismus

**Polymorphismus** := Fähigkeit einer *Variablen*, zur Laufzeit auf Objekte von beliebigem Typ verweisen zu können

**Polymorphismus** := Fähigkeit einer *Prozedur*, Argumente verschiedener Typen entgegennehmen zu können

## in Visual Basic

⇒ allgemeiner Objekttyp 'Object'

⇒ 'Variant'-Datentyp

## Beispiel

cTestA
<pre>Public Sub Zeige()     MsgBox "Typ cTestA" End Sub</pre>

cTestB
<pre>Public Sub Zeige()     MsgBox "Typ cTestB" End Sub</pre>

## Testprozedur

```
Dim Obj As Object
```

```
If InputBox("Wählen Sie A oder B") = "A" Then
```

```
    Set Obj = New cTestA
```

```
Else
```

```
    Set Obj = New cTestB
```

```
End If
```

```
Obj.Zeige
```

← zeige oder zeige ?

# Vererbung

**Vererbung** := Konzept, bei dem eine neue Klasse das Verhalten und die Struktur bestehender Klassen übernehmen (erben) und ändern kann, ohne sie neu entwerfen und implementieren zu müssen

⇒ Klassifikation

⇒ Klassifikationshierarchien

## in Visual Basic

⇒ bis Office97 nur *erben* von Funktionalität möglich

⇒ ab Office2000 auch Vererbung möglich

**Beispiel** ([grüsseHöflich.xls](#) ➔ Erben)

UserForm	
Left	UserForm1
Visible	
...	
Hide	Move
Show	...



frmGrüsse	
Left	Begrüssung
Visible	lblBegrüssung
Gruss	
...	
Hide	Move
Show	...

**Private** Dialog **As** frmGrüsse

**Sub** grüsseHöflich()

**Set** Dialog = **New** frmGrüsse

    Dialog.Gruss = "Guten Tag"

    Dialog.Show

**End Sub**

# Datenverwaltung

---

## Daten verwalten heisst

- **speichern**
  - ⇒ schnell und platzsparend
- **wiederfinden**
  - ⇒ schnell und zuverlässig
- entfernen, sortieren, kopieren, ...

## Beispiel: Datenfeld

- **speichern**
- **wiederfinden**
  - ⇒ direkte Suche
  - ⇒ sequentielle Suche
  - ⇒ binäre Suche

## Beispiel: Bibliothek, Bücher

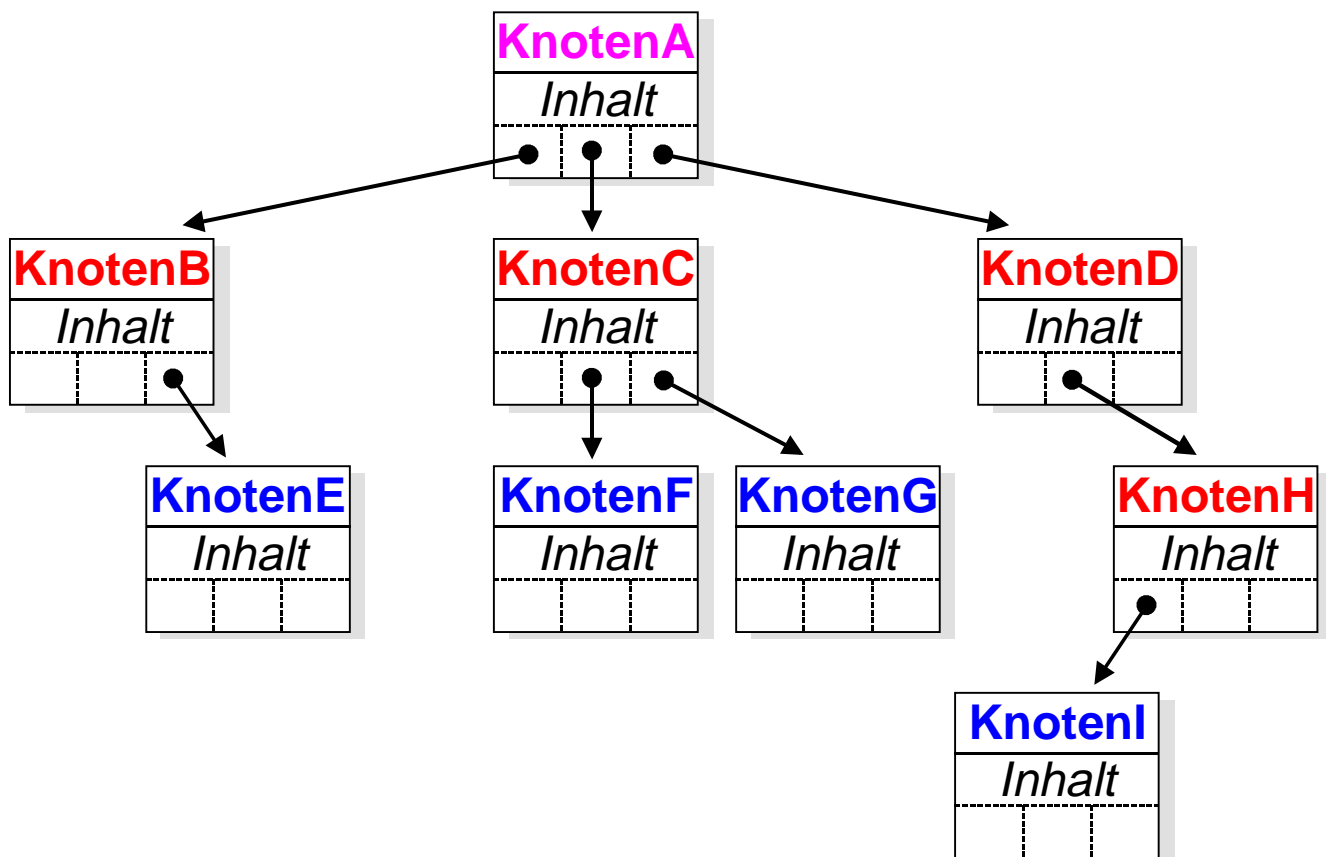
- **speichern**
- **wiederfinden**
  - ⇒ Katalog
  - ⇒ Inhaltsverzeichnis
  - ⇒ Index



# Baum-Strukturen

Ein **Baum** ist eine kreislos verkettete Datenstruktur, wobei jedes Element auf *mehrere* Nachfolger verweisen kann.

- **verkettete** Datenstruktur
- **nicht-lineare** Datenstruktur
- die Elemente heissen **Knoten**, **Wurzel**, **Blätter**, **Eltern**, **Kinder**
- legt man für die Nachfolger eine Reihenfolge fest, so heisst der Baum *geordnet* (Achtung: geordnet  $\neq$  sortiert)
- Spezialfälle: **verkettete Liste**, **Binärbaum**



# verkettete Datenstrukturen

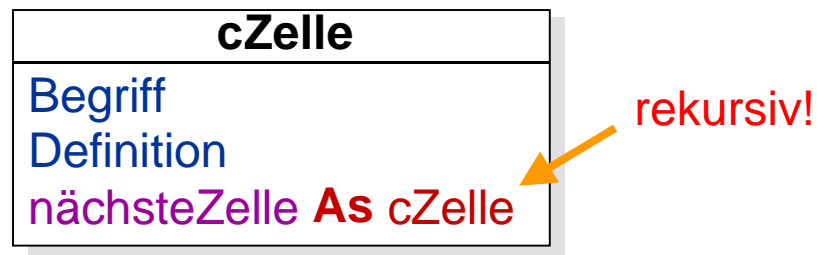
## Datenstruktur

Die Elemente bestehen aus zwei Teilen:

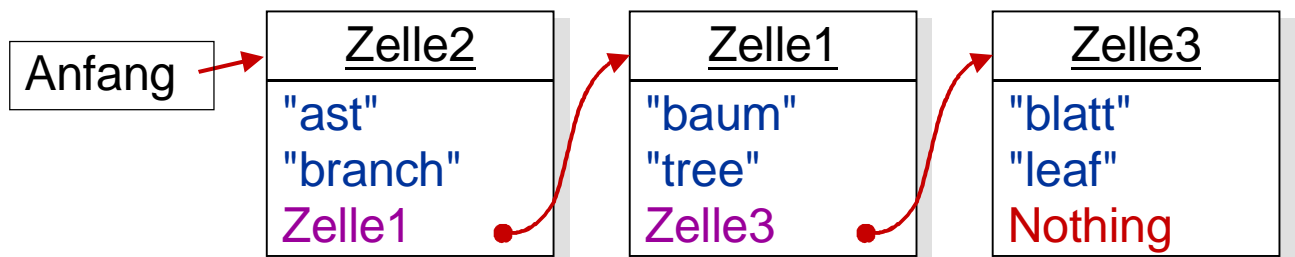
- 1) **Inhalt**
- 2) **Verweis(e)** auf andere Elemente (Verkettung)

## Beispiel: verkettete Liste

Element (Zelle)



Liste



## Besonderheiten

- ☺ dynamisch
- ☺ keine Reihenfolge im Speicher
- ☺ kein Kopieren von Inhalten bei Umstrukturierung (z.B. einfügen)
- ☹ keine direkte Suche

# Implementation der verketteten Liste

## Klassenmodul cZelle

```
Public Inhalt As String
Public Nachfolger As cZelle
```

## Codemodul

```
Private Anfang As cZelle
```

```
Private Sub Test()
```

```
Dim Zelle1 As cZelle
Dim Zelle2 As cZelle
Dim Zelle3 As cZelle
Dim Zelle4 As cZelle
```

*Variablen  
vereinbaren*

```
Set Zelle1 = New cZelle
Zelle1.Inhalt = "anna"

Set Zelle2 = New cZelle
Zelle2.Inhalt = "carmen"

Set Zelle3 = New cZelle
Zelle3.Inhalt = "david"

Set Zelle4 = New cZelle
Zelle4.Inhalt = "beat"
```

*Elemente  
erstellen*

```
Set Anfang = Zelle1
Set Zelle1.Nachfolger = Zelle2
Set Zelle2.Nachfolger = Zelle3
```

*Elemente  
verketten*

```
fügeEin Zelle4, Zelle1
```

*Element  
einfügen*

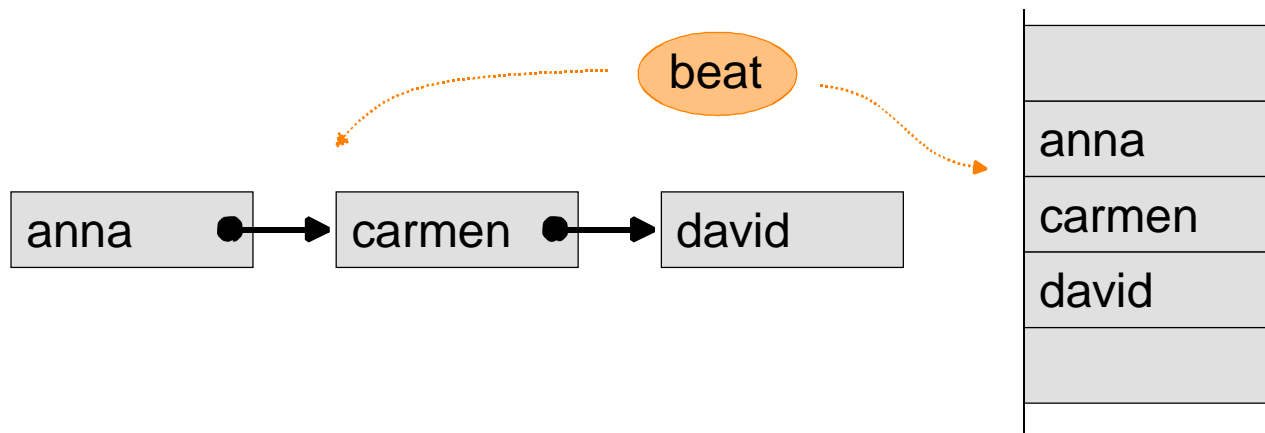
```
End Sub
```

```
Private Sub fügeEin(neu As cZelle, hinter As cZelle)
```

```
Set neu.Nachfolger = hinter.Nachfolger
Set hinter.Nachfolger = neu
```

```
End Sub
```

# Vergleich verkettete Liste - Datenfeld



## verkettete Liste

## Datenfeld

### Einfügen

unsortiert:

😊 einfach und schnell

😞 schwierig

sortiert:

😊 einfach (und schnell)

😞 schwierig

### Suchen

direkt:

😞 nicht möglich

😊 schnell

sequentiell:

😞 langsam

😞 langsam

binär:

😞 nicht möglich

😊 schnell

### Löschen

😊 einfach und schnell,  
wenn *doppelt verkettet*

😞 schwierig

Strukturen, die auch Rückwärtsverweise verwalten,  
bezeichnet man als *doppelt verkettet*.

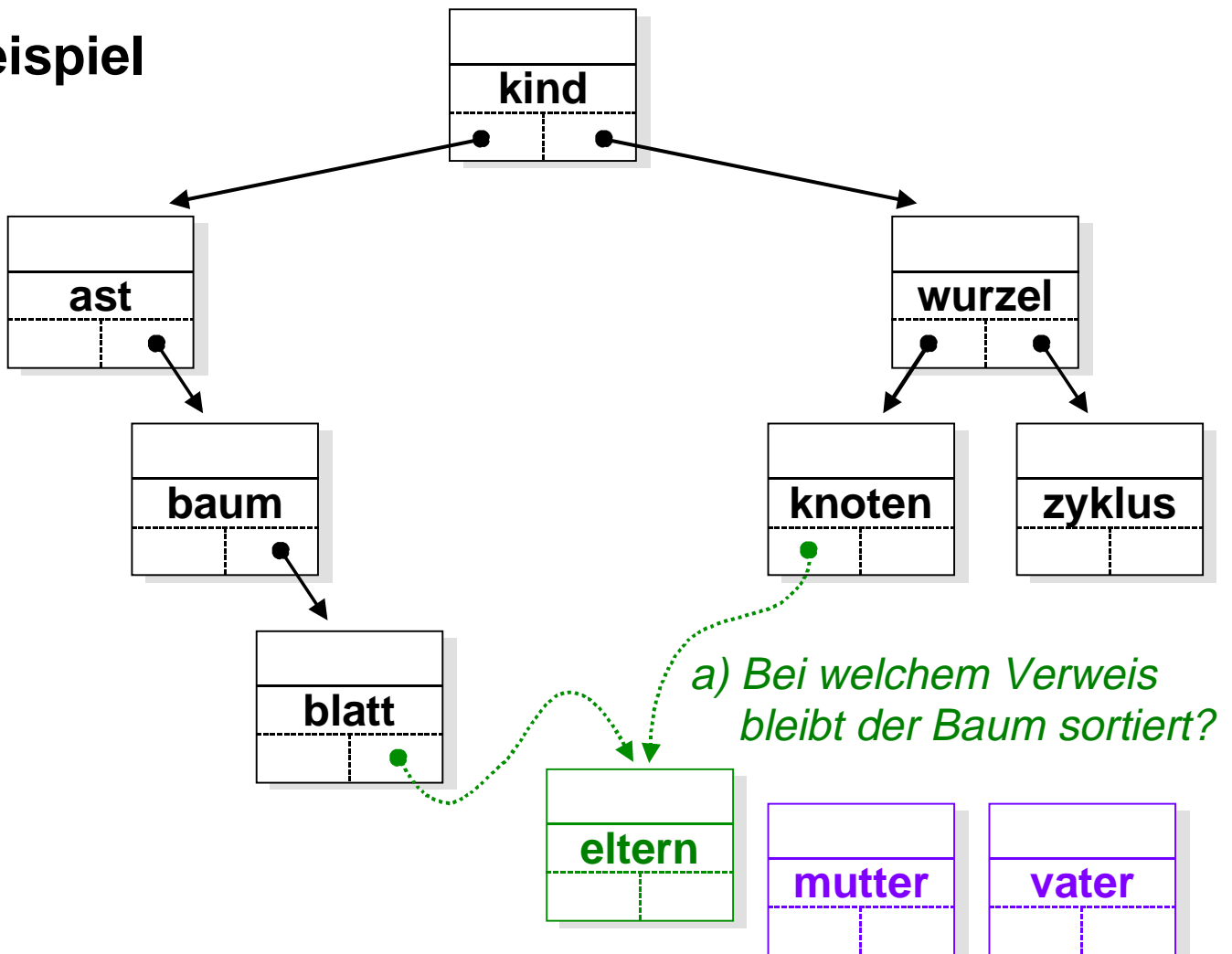
# Binärbaum

Ein **Binärbaum** ist ein *geordneter* Baum, dessen Knoten *zwei* Nachfolger haben, einen *linken* und einen *rechten*.

Ein Binärbaum heisst **sortiert**, wenn *für jeden* Knoten *k* gilt:

1. im *linken* Unterbaum sind alle Inhalte kleiner als in *k*  
und
2. im *rechten* Unterbaum sind alle Inhalte grösser als in *k*.

## Beispiel



a) Bei welchem Verweis bleibt der Baum sortiert?

b) Fügen Sie die anderen zwei Knoten so ein, dass der Baum sortiert bleibt?

c) Formulieren Sie einen Einfügealgorithmus.

# Einfügealgorithmus (iterativ)

## Entwurf

**fügeHinzult**(Begriff<sup>(1)</sup>)

FALLS Baum noch keinen Wurzelknoten hat  
erzeuge Wurzelknoten  
fülle Begriff ein.

SONST

gehe zu Wurzelknoten

WIEDERHOLE BIS Begriff = Begriff im Knoten

FALLS Begriff < Begriff im Knoten

FALLS Knoten keinen linken Nachfolger hat  
erzeuge linken Nachfolgerknoten  
fülle Begriff ein.

gehe zu linkem Nachfolgerknoten

SONST

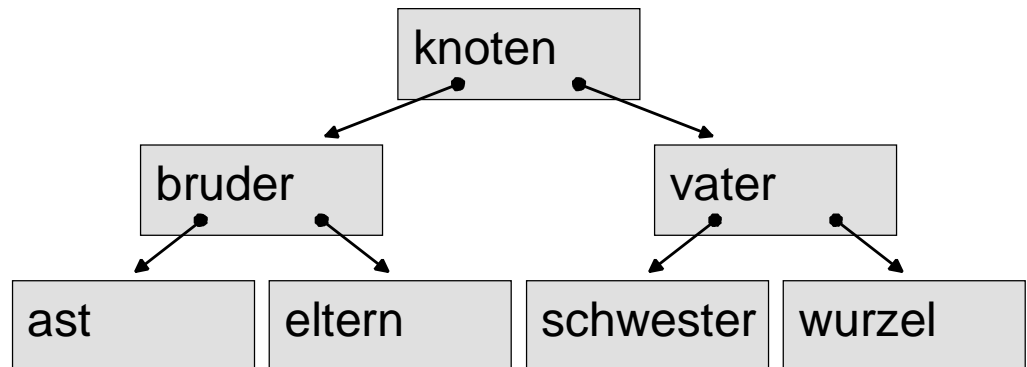
FALLS Knoten keinen rechten Nachfolger hat  
erzeuge rechten Nachfolgerknoten  
fülle Begriff ein.

gehe zu rechtem Nachfolgerknoten

<sup>(1)</sup> Begriff steht für das Begriff/Definitions-Paar

# Suche im sortierten Binärbaum

ast
bruder
eltern
knoten
schwester
vater
wurzel



## Beispiele

- suche 'eltern'
- suche 'vater'

## Vergleich Binärbaum

Einfügen	😊 schnell
Suche	😊 schnell wie Binär- suche im sortierten Datenfeld ( <i>im Idealfall !</i> )

## verkettete Liste

😊 einfach und schnell
😞 sequentiell, langsam

# Suchalgorithmus (iterativ)

## Entwurf

**suchelt**(Begriff<sup>(1)</sup>)

FALLS Baum keinen Wurzelknoten hat  
gib **Nothing**<sup>(2)</sup> zurück.

SONST

gehe zu Wurzelknoten

WIEDERHOLE BIS **Begriff** = Begriff im aktuellen Knoten

FALLS **Begriff** < Begriff im aktuellen Knoten

FALLS aktueller Knoten keinen **linken** Nachfolger hat  
gib **Nothing**<sup>(2)</sup> zurück.

gehe zu **linkem** Nachfolgerknoten

SONST

FALLS aktueller Knoten keinen **rechten** Nachfolger hat  
gib **Nothing**<sup>(2)</sup> zurück.

gehe zu **rechtem** Nachfolgerknoten

gib **aktuellen Knoten** zurück.

<sup>(1)</sup> **Begriff** steht für das Begriff/Definitions-Paar

<sup>(2)</sup> **Nothing** wird zurück gegeben, falls gesuchter **Begriff** nicht existiert



---

# Der “ideale Binärbaum” ?

---





## Aufgaben

Skizzieren Sie die Baumstruktur, die entsteht, wenn Sie die folgenden Begriff in einen (noch leeren) sortieren Binärbaum einfüllen:

- a) *kind, baum, blatt, wurzel, zyklus, knoten, ast*
- b) *ast, baum, blatt, kind, knoten, wurzel, zyklus*

## Fragen

- 1) Wenn Sie in den obigen Beispielen den letzten Begriff einfüllen, wie oft müssen Sie ihn dann mit bereits im Baum vorhandenen Begriffen vergleichen?
  - a) .....
  - b) .....
- 2) Wie viele Begriffe kann ein sortierter Binärbaum im Idealfall enthalten, so dass man jeden weiteren Begriff mit maximal 8 Vergleichen hinzufügen kann?

  
.....  

- 3) Unter welchen Bedingungen tritt der ungünstigste Fall ein?  
.....  
  
  


# Höhe und Ausbalanciertheit

Als **Höhe** eines Baumes bezeichnet man die Anzahl Verweise, denen man maximal folgen muss, um zu einem Blatt zu gelangen.

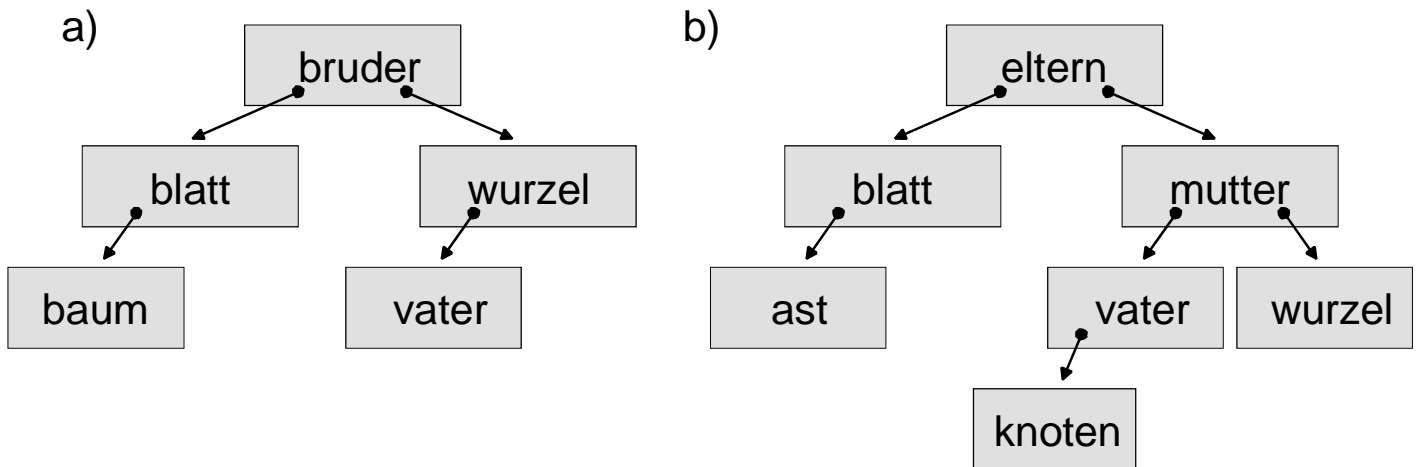
Je geringer die Höhe, desto effizienter die Algorithmen.

Die Höhe sortierter Binärbäume hängt davon ab,

- *wie viele* Knoten der Baum speichern muss und
- *wie ausbalanciert* die Teilbäume der einzelnen Knoten sind.

Ein Baum heisst **ausbalanciert**, wenn *für jeden* Knoten gilt, dass sich die Anzahl der Knoten in den beiden Unterbäumen höchstens um Eins unterscheiden.

# Beispiele



## Fragen:

1) Welche Höhe haben die beiden Bäume?

a) .....

b) .....

2) Welcher Baum ist sortiert, welcher nicht?

a) .....

b) .....

3) Welcher Baum ist ausbalanciert, welcher nicht?

a) .....

b) .....



# rekursive Datenstrukturen

## alternative Definition

Ein **Baum** ist entweder ...

1. ein Knoten ohne Nachfolger ( $\rightarrow$ Blatt)      *oder*
2. ein Knoten mit **Bäumen** als Nachfolger ( $\rightarrow$ Teilbäume)

$\Rightarrow$  rekursive Definition

$\Rightarrow$  Auf rekursiv definierten Datenstrukturen lassen sich oft auch einfache rekursive Algorithmen formulieren.

## zur Erinnerung

- Die Rekursion ist eine Form der *Wiederholung*.
- Eine alternative Form der Wiederholung ist die *Iteration*.
- Man unterscheidet zwischen *endlicher* und *unendlicher* Rekursion.
- Man unterscheidet zwischen *direkter* und *indirekter* Rekursion.

---

# rekursive Algorithmen

---

## Konstruktion rekursiver Algorithmen

1. **Basisfall** := Fall, der ohne rekursiven Aufruf gelöst werden kann. Der Basisfall bricht die Rekursion ab (→ Abbruchbedingungen).

z.B.: Blatt

2. **Reduktion** := reduziert die gestellte Aufgabe auf eine einfachere und lässt diese lösen (→ rekursive Aufrufe).

z.B.: Teilbaum

Damit ein Algorithmus abbricht, muss ...

- ... der Basisfall *vor* der Reduktion geprüft werden
- ... die Reduktion die gestellte Aufgabe *vereinfachen*

# Einfügealgorithmus (rekursiv)

## Entwurf

**fügeHinzuRek**(Begriff<sup>(1)</sup>, Baum<sup>(2)</sup>)

'--- *Basisfall*

FALLS Baum leer ist  
    erzeuge Wurzelknoten  
    fülle Begriff ein.

'--- *Reduktionen*

SONST

FALLS Begriff < Begriff im Wurzelknoten von Baum  
    **fügeHinzuRek** Begriff, linker Teilbaum des Baumes

FALLS Begriff > Begriff im Wurzelknoten von Baum  
    **fügeHinzuRek** Begriff, rechter Teilbaum des Baumes

<sup>(1)</sup> Begriff steht wieder für das Begriff/Definitions-Paar

<sup>(2)</sup> Teilbaum, in dem der Begriff hinzugefügt werden soll

Wann rekursiv, wann iterativ ?

## Aufgabe

Wir möchten ein Wörterbuch 'Deutsch-Englisch' für Fachbegriffe implementieren.

Später möchten wir weitere Wörterbücher erstellen (z.B. 'Englisch-Deutsch').

## Benutzerschnittstelle

deutsch	englisch
kind	child
<input type="button" value="füge hinzu"/>	<input type="button" value="übersetze"/>
<input type="button" value="entferne"/>	

- **füge hinzu:** Ordnet einen neuen *Eintrag* (*Begriff/Definitions-Paar*) ins Wörterbuch ein.
- **übersetze:** Gibt Definition zu einem Begriff zurück.
- **entferne:** Entfernt einen Eintrag aus dem Wörterbuch.

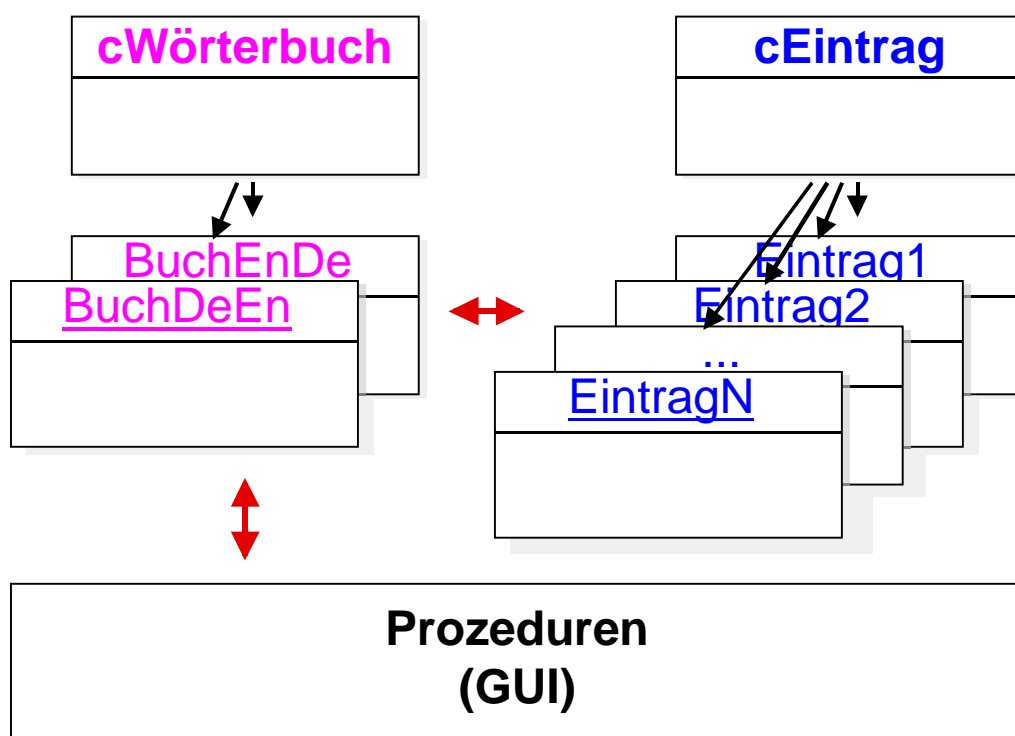
## Lösungsvarianten

- ☹ Wörterbuch erstellen und später kopieren
- ☹ gemeinsamen Code in Prozeduren auslagern
- 😊 Wörterbuch in Klassenmodul implementieren

# Architektur

## Module

- Klassenmodul **cEintrag**  
⇒ implementiert die Knoten der Bäume
- Klassenmodul **cWörterbuch**  
⇒ verwaltet Wörterbuch-Einträge in einem Binärbaum
- Standardmodul Prozeduren  
⇒ enthält Ereignisprozeduren der Benutzerschnittstelle





# Klasse cEintrag

cEintrag	
Begriff	Definition
Vater	
KindLi	KindRe

## Eigenschaften (und Methoden)

Eigenschaft	Datentyp	Beschreibung
Begriff	String	Schlüssel
Definition	String	Inhalt
Vater	cEintrag	Verkettung zu Vorgänger
KindLi	cEintrag	Verkettung zu linkem Nachfolger
KindRe	cEintrag	Verkettung zu rechtem Nachfolger

Verweise auf Vater sind zwar redundant, vereinfachen aber die Algorithmen.

## Implementation

Klassenmodul cEintrag

```
Public Begriff As String
Public Definition As String
Public KindLi As cEintrag
Public KindRe As cEintrag
Public Vater As cEintrag
```

rekursive Definition

doppelt verkettet

# Klasse cWörterbuch

## Eigenschaften und Methoden

(in Analogie zum Collection-Objekt)

Methoden	Collection	Beschreibung
fügeEin(Begriff, Definition)	Add	ordnet Eintrag ein
Eintrag(Begriff)	Item	gibt Definition zurück
entferne(Begriff)	Remove	entfernt Eintrag
OEster() ONächster(Eintrag)	For Each ... Next	erster Eintrag nächster Eintrag

- cWörterbuch sortiert die Objekte, Collection nicht
- cWörterbuch kann nur Objekte vom Typ cEintrag aufnehmen
- Wir können keine For Each...Next-Schleife implementieren

# Verwenden von cWörterbuch

## Beispiel

```
Dim BuchDeEn As cWörterbuch
```

```
Dim Eintrag As cEintrag
```

```
Set BuchDeEn = New cWörterbuch
```

```
BuchDeEn.fügeHinzu "baum", "tree"
```

```
BuchDeEn.fügeHinzu "ast", "branch"
```

```
BuchDeEn.fügeHinzu "kind", "child"
```

```
Set Eintrag = BuchDeEn.OErster()
```

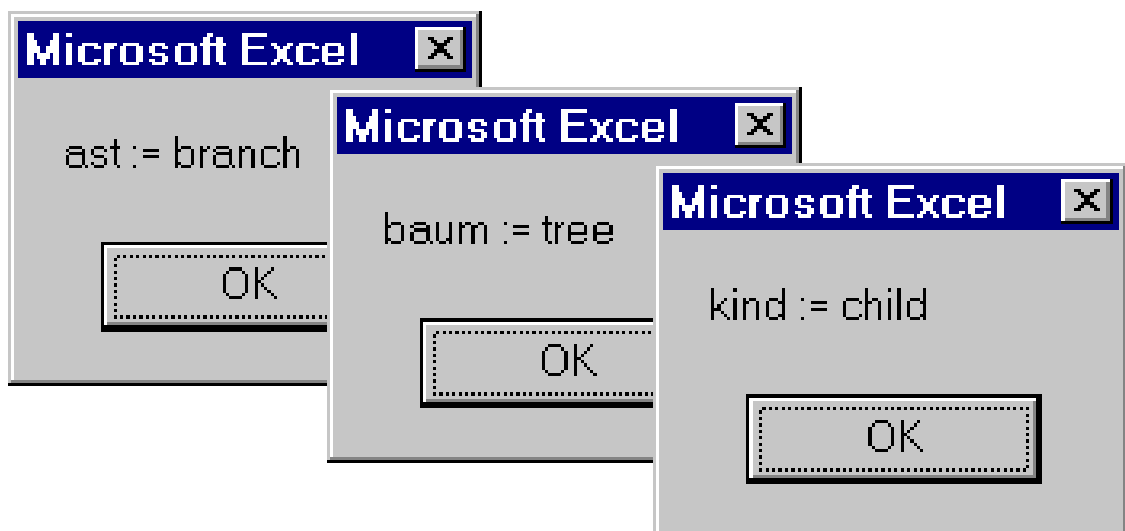
```
Do While Not Eintrag Is Nothing
```

```
    MsgBox Eintrag.Begriff & " := " & Eintrag.Definition
```

```
    Set Eintrag = BuchDeEn.ONächster(Eintrag)
```

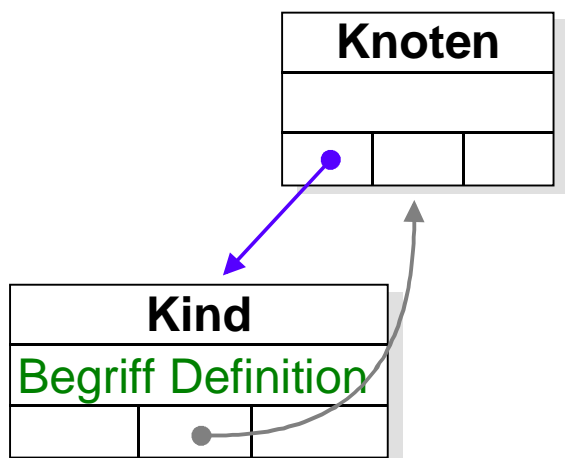
```
Loop
```

## Ausgabe:

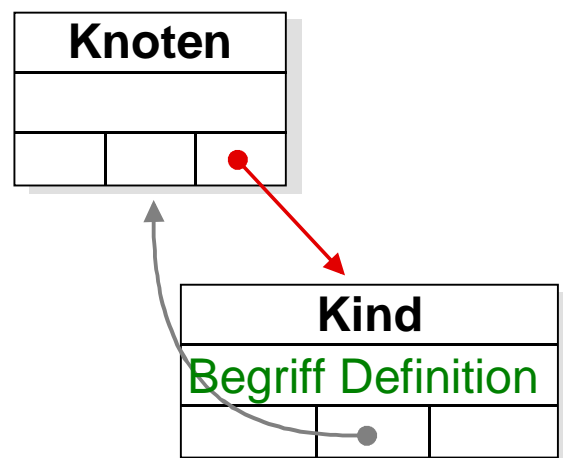


# Implementation der Verkettung

erzeugeKindLi( )



erzeugeKindRe( )



## im Klassenmodul cWörterbuch

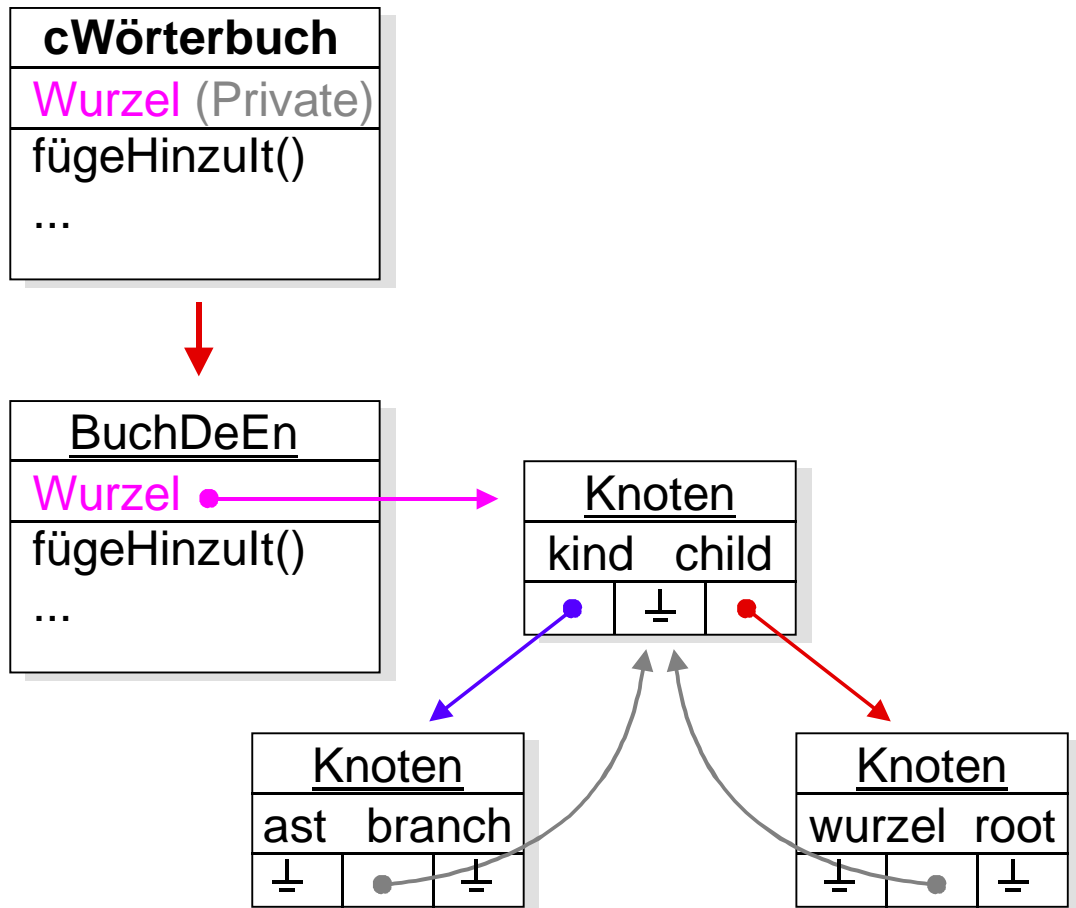
```
Private Sub erzeugeKindLi(Knoten As cEintrag, _  
                           Begriff As String, _  
                           Definition As String)  
  
    Set Knoten.KindLi = New cEintrag  
    Set Knoten.KindLi.Vater = Knoten  
  
    Knoten.KindLi.Begriff = Begriff  
    Knoten.KindLi.Definition = Definition  
  
End Sub
```

```
Private Sub erzeugeKindRe(Knoten As cEintrag, _  
                           Begriff As String, _  
                           Definition As String)  
  
    .....  
    .....  
    .....  
    .....  
  
End Sub
```



# Implementation der Baum-Struktur

## Klasse cWörterbuch



---

# Einfügealgorithmus (iterativ)

---

## Implementation im Klassenmodul cWörterbuch

```
Private Wurzel As cEintrag

Public Sub fügeHinzuIt(Begriff As String, _
                      Definition As String)

    Dim Knoten As cEintrag

    If Wurzel Is Nothing Then
        Set Wurzel = New cEintrag
        Wurzel.Begriff = Begriff
        Wurzel.Definition = Definition
    Else
        Set Knoten = Wurzel
        Do Until Begriff = Knoten.Begriff
            If Begriff < Knoten.Begriff Then
                If Knoten.KindLi Is Nothing Then
                    erzeugeKindLi Knoten, Begriff, Definition
                End If
                Set Knoten = Knoten.KindLi
            Else
                If Knoten.KindRe Is Nothing Then
                    erzeugeKindRe Knoten, Begriff, Definition
                End If
                Set Knoten = Knoten.KindRe
            End If
        Loop
    End If
End Sub
```

---

# Einfügealgorithmus (rekursiv)

---

## Implementation im Klassenmodul cWörterbuch

```
Private Sub fügeHinzuRek(Begriff As String, _  
                        Definition As String, _  
                        Baum As cEintrag)
```

```
    'Abbruchbedingungen:
```

```
    If Wurzel Is Nothing Then
```

```
        Set Wurzel = New cEintrag
```

```
        Wurzel.Begriff = Begriff
```

```
        Wurzel.Definition = Definition
```

```
    ElseIf Begriff < Baum.Begriff _
```

```
    And Baum.KindLi Is Nothing Then
```

```
        erzeugeKindLi Baum, Begriff, Definition
```

```
    ElseIf Begriff > Baum.Begriff _
```

```
    And Baum.KindRe Is Nothing Then
```

```
        erzeugeKindRe Baum, Begriff, Definition
```

```
    'Reduktionen:
```

```
    ElseIf Begriff < Baum.Begriff Then
```

```
        fügeHinzuRek Begriff, Definition, _  
                    Baum.KindLi
```

```
    ElseIf Begriff > Baum.Begriff Then
```

```
        fügeHinzuRek Begriff, Definition, _  
                    Baum.KindRe
```

```
    End If
```


```
End Sub
```

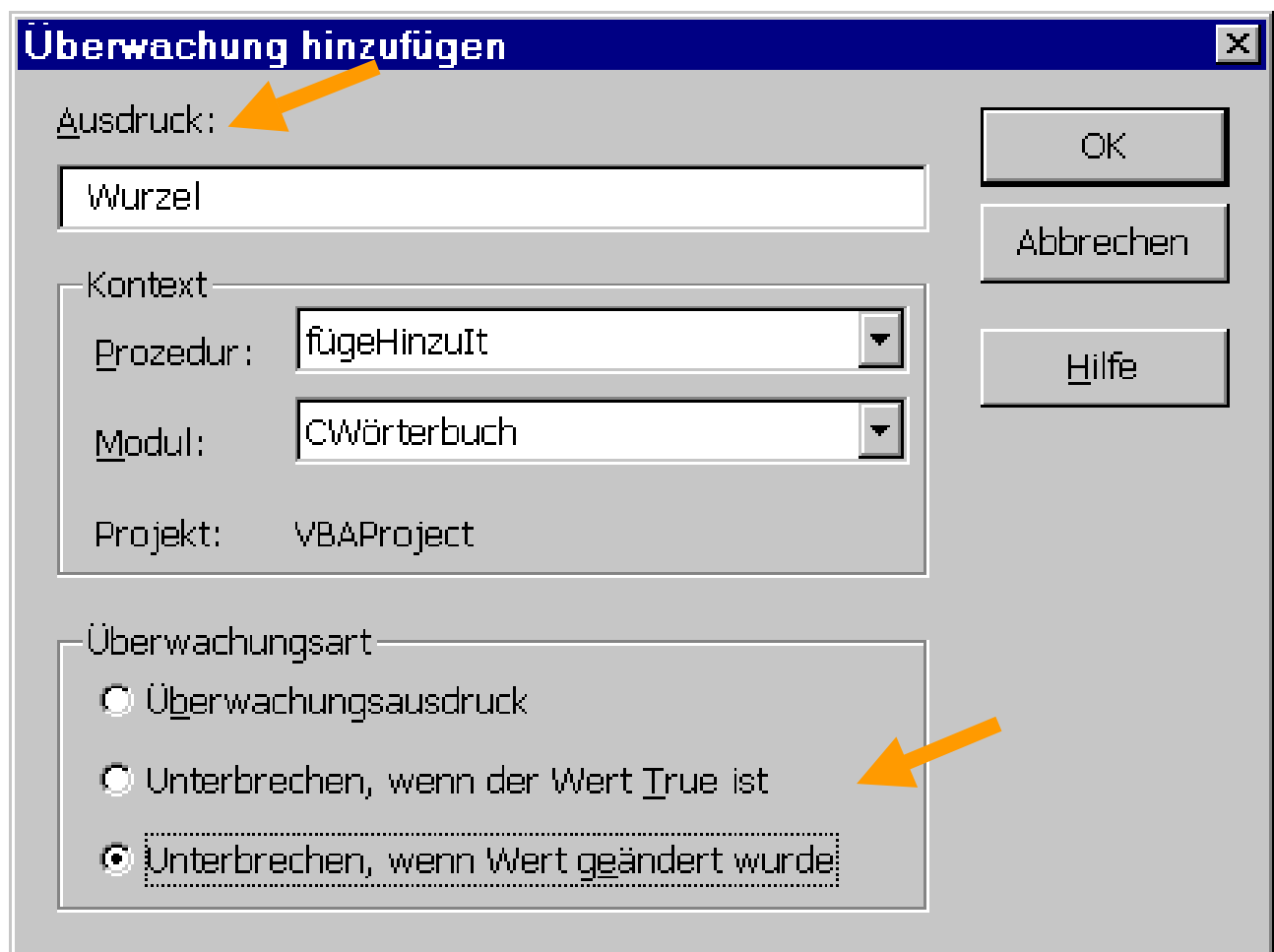
## Aufruf

```
fügeHinzuRek Begriff, Definition, Wurzel
```

# Test mit VBA-Überwachungsausdrücken

## Überwachungsausdruck hinzufügen

- 1) Überwachung hinzufügen 
- 2) Überwachungsausdruck definieren; ev. Haltepunkt setzen



- 3) Ausdruck im Überwachungsfenster beobachten

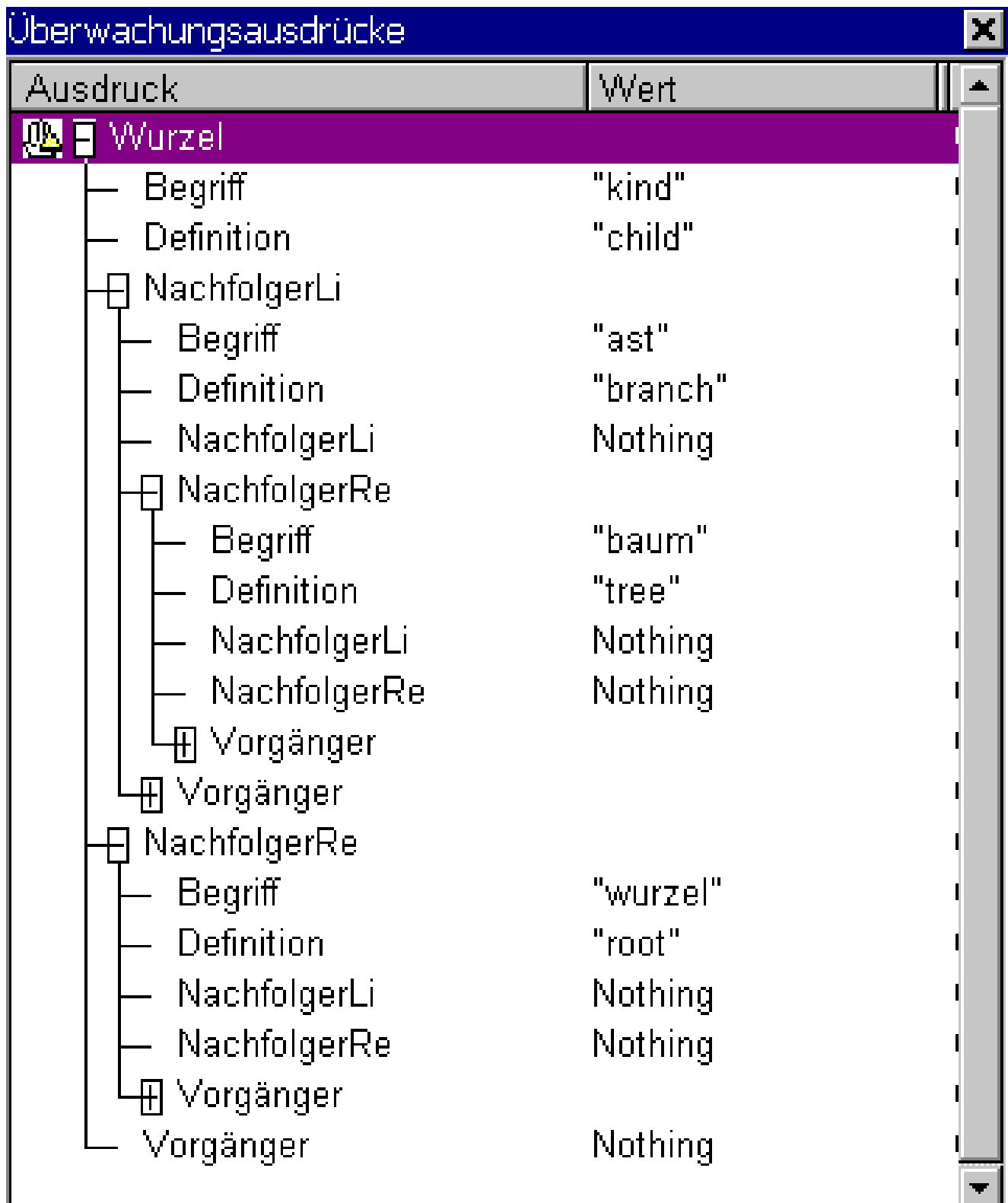


Ausdruck	Wert	Typ	Kontext
 Wurzel	<Nicht im Kontext>	CEintrag	CWörterbuch.fügeHinzuIt



# Baumstruktur im Überwachungsfenster

nach Hinzufügen der Begriffe *kind*, *ast*, *baum*, *wurzel*



---

# 6. Woche

---

## Vorlesung: Bäume

Repetition: Datenverwaltung, Bäume, Binärbäume

Implementation in Visual Basic (Übung Woerterbuch)

Binärbäume

- der ideale Baum (Höhe und Ausbalanciertheit)

Vergleich Datenstrukturen

Bäume in Datenbanken

Algorithmen

- Rekursion

  - rekursive Datenstrukturen

  - rekursive Algorithmen

  - rekursiver Einfügealgorithmus

- Baumtraversierungen

  - Tiefentraversierung

    - Präordnung-Tiefentraversierung

    - Postordnung-Tiefentraversierung

    - Inordnung-Tiefentraversierung

      - erstes Element finden

      - nächstes Element finden

  - Breitentraversierung

    - erstes Element finden

    - nächstes Element finden

## Übung: WoerterbuchStandard

---

## 7. Woche

---

**keine Vorlesung (Pfingsten)**

**Übung: WoerterbuchProfessional**

# Datenverwaltung

Daten verwalten heisst

- **speichern**
  - ⇒ schnell und platzsparend
- **wiederfinden**
  - ⇒ schnell und zuverlässig
- entfernen, sortieren, kopieren, ...

Beispiele: Bücher, Explorer, Übung

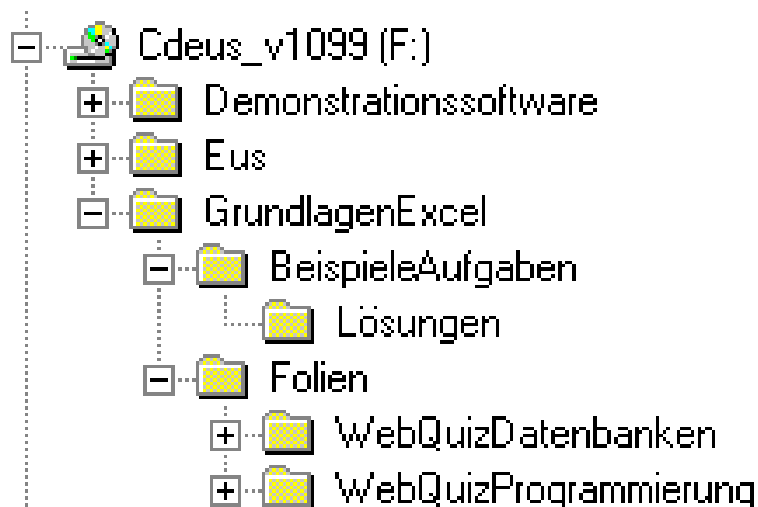
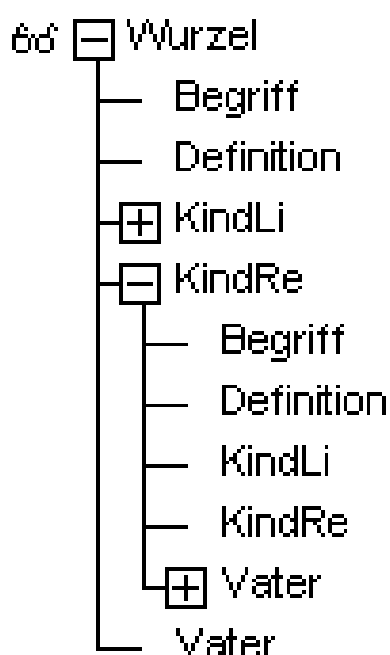
## 3 Bäume 57

3.1 Datenverwaltung ..... 58

3.2 Baumstrukturen ..... 59

3.2.1 Definitionen ..... 59

3.2.2 Beispiel Wörterbuch.. 60



## Stichwortverzeichnis

### A

Ablaufstruktur 80,81,83

absolute Adresse 24

# Baum und Binärbaum

Ein **Binärbaum** ist ein *geordneter* Baum, dessen Knoten *zwei* Nachfolger haben, einen *linken* und einen *rechten*.

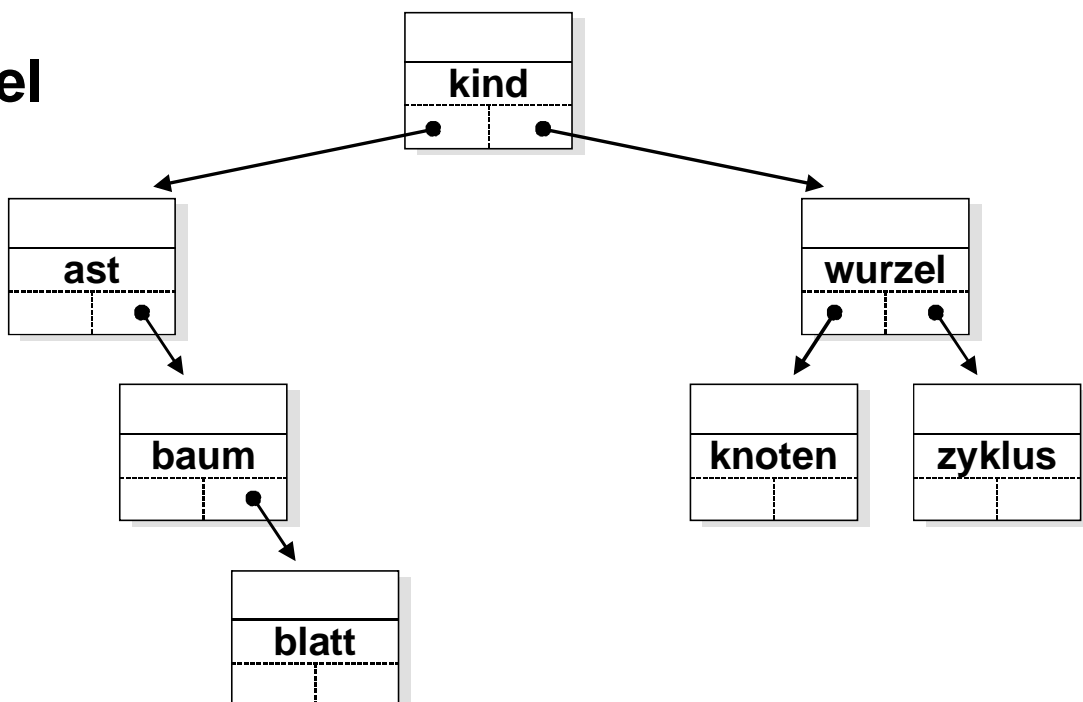
Ein Binärbaum heisst **sortiert**, wenn *für jeden* Knoten  $k$  gilt:  
1. im *linken* Unterbaum sind alle Inhalte kleiner als in  $k$  und  
2. im *rechten* Unterbaum sind alle Inhalte grösser als in  $k$ .

Die Anzahl Verweise, denen man maximal folgen muss, um zu einem Blatt zu gelangen, heisst **Höhe** eines Baumes.

Ein Baum heisst **ausbalanciert**, wenn *für jeden* Knoten gilt, dass sich die Anzahl der Knoten in den beiden Unterbäumen höchstens um Eins unterscheiden.

- **verkettete** Datenstruktur  $\Rightarrow$  vereinfacht das Einfügen
- **nicht-lineare** Datenstruktur  $\Rightarrow$  beschleunigt die Suche
- Je flacher ein Baum, desto schneller die Algorithmen

## Beispiel



# Vergleich Datenstrukturen

	Daten- feld	verkettete Liste	Binär- baum
--	----------------	---------------------	----------------

## Besonderheiten

statisch <sup>(1)</sup>	dynamisch	dynamisch
zusammen	verkettet	verkettet
linear	linear	nicht-linear

## Algorithmen

unsortiert einfügen	aufwändig <sup>(1)</sup>	einfach	einfach
sortiert einfügen	aufwändig <sup>(1)</sup>	aufwändig	einfach
suche binär <sup>(2)</sup>	schnell	-	schnell <sup>(3)</sup>
suche sequentiell	langsam	langsam	langsam

## Anmerkungen

(1) es gibt auch dynamische Datenfelder

(2) nur wenn Daten sortiert

(3) nur wenn Baum ausbalanciert

# Bäume und Datenbanken

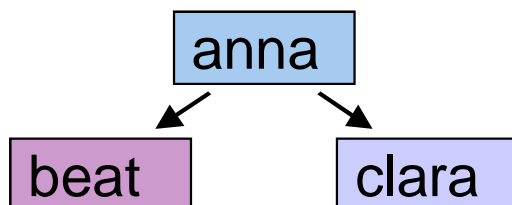
## Datei

	anna	302 33 12	309	clara	267 45 21	107	beat	267 ...
--	------	-----------	-----	-------	-----------	-----	------	---------

## Tabelle (Datenbank)

<i>Name</i>	<i>Telefon</i>	<i>Büro</i>
anna	302 33 12	309
clara	267 45 21	107
beat	267 45 34	108

## Index



- 😊 Ein Index beschleunigt die Suche in Daten,
- 😞 verlangt jedoch zusätzlichen Speicherplatz und
- 😞 verlangsamt Schreib- und Löschoperationen.

## Aufgabe

Wir möchten ein Wörterbuch 'Deutsch-Englisch' für Fachbegriffe implementieren. Später möchten wir weitere Wörterbücher erstellen (z.B. 'Englisch-Deutsch').

## Benutzerschnittstelle

<i>deutsch</i>	<i>englisch</i>
<input type="text" value="baum"/>	<input type="text" value="tree"/>
<input type="button" value="füge hinzu"/>	<input type="button" value="übersetze"/> <input type="button" value="entferne"/>

<i>listeDeutsch</i>	<i>listeEnglisch</i>
ast	branch
baum	tree
blatt	leaf
kind	child
knoten	node
wurzel	root

## Lösungsvarianten

- ☹ Wörterbuch erstellen und später kopieren
- ☹ gemeinsamen Code in Prozeduren auslagern
- 😊 Wörterbuch in Klassenmodul implementieren

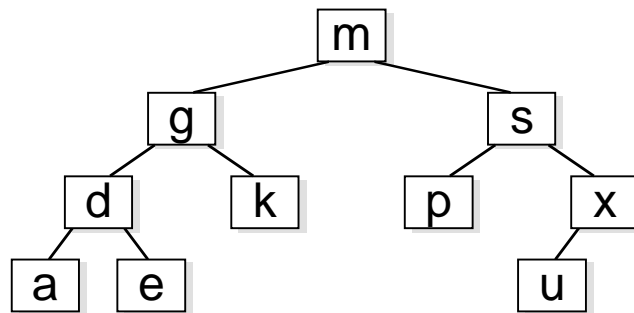


# Baumtraversierungen

Einen Baum **traversieren** heisst ausgehend von der Wurzel, alle Knoten besuchen und jeden einmal ausgeben.

## Beispiele

sortierter Binärbaum:



a) in sortierter Reihenfolge:  
a, d, e, g, k, m, p, s, u, x

b) ebenenweise:  
m, g, s, d, k, p, x, a, e, u

## Traversierungsarten

Eine **Tiefentraversierung** besucht zunächst die Kinder und erst dann den Ebenennachbar eines Knotens.

Eine **Breitentraversierung** besucht zunächst den Ebenennachbar eines Knotens und erst dann die Kinder.

# Tiefentraversierung

## Verschiedene Tiefentraversierungen

### Präordnungs-Traversierung

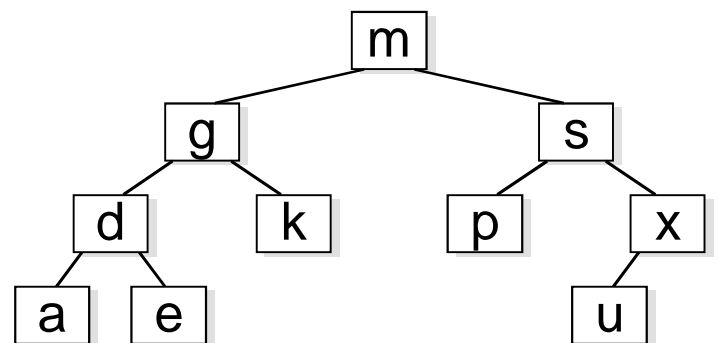
1. besuche Vater
2. besuche alle Kinder

### Inordnungs-Traversierung (nur bei Binärbäumen)

1. besuche linkes Kind
2. besuche Vater
3. besuche rechtes Kind

### Postordnungs-Traversierung

1. besuche alle Kinder
2. besuche Vater



## Beispiele

sortierter Binärbaum:

a) Präordnungs-Traversierung:

.....

b) Inordnungs-Traversierung:

.....

c) Postordnungs-Traversierung



.....

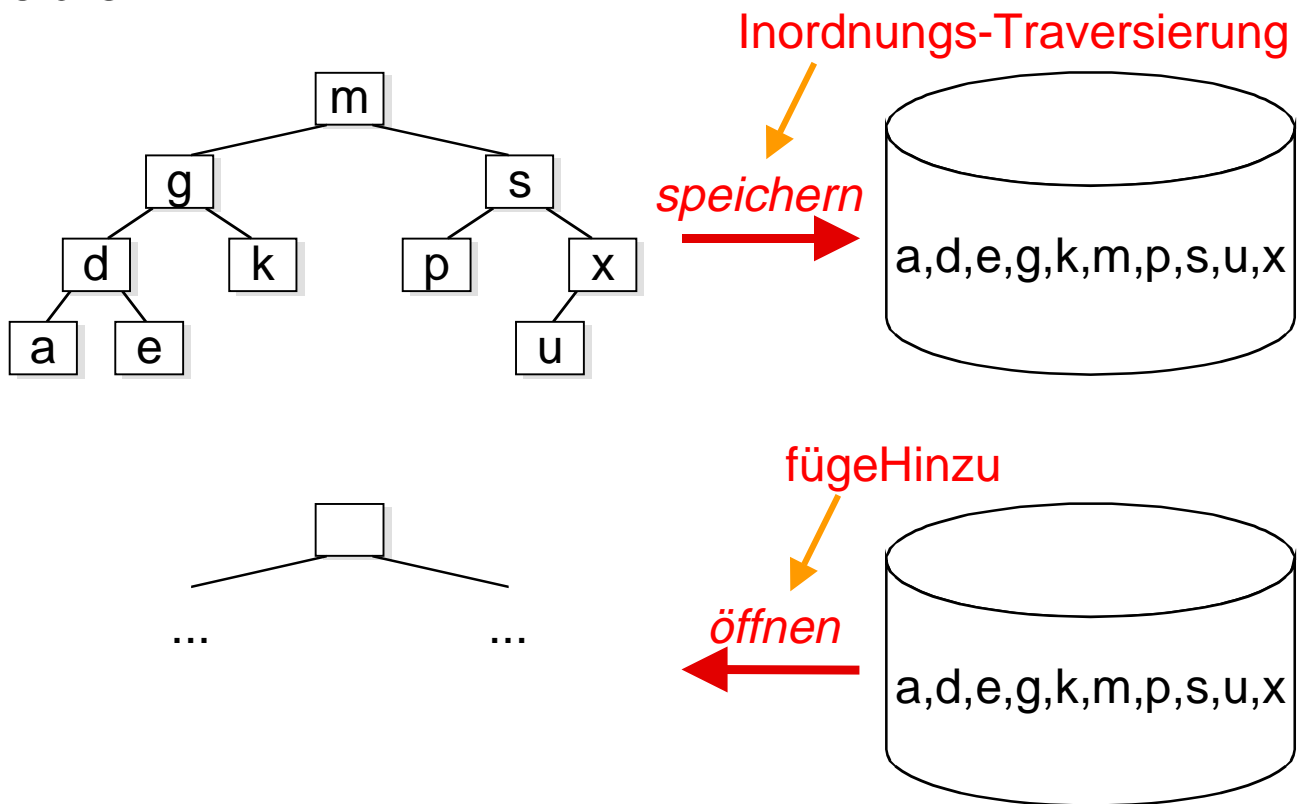
Welche Traversierung eignet sich zur Ausgabe einer nach der deutschen Begriffen (Schlüssel) sortierten Liste der Einträge unseres Wörterbuchs?



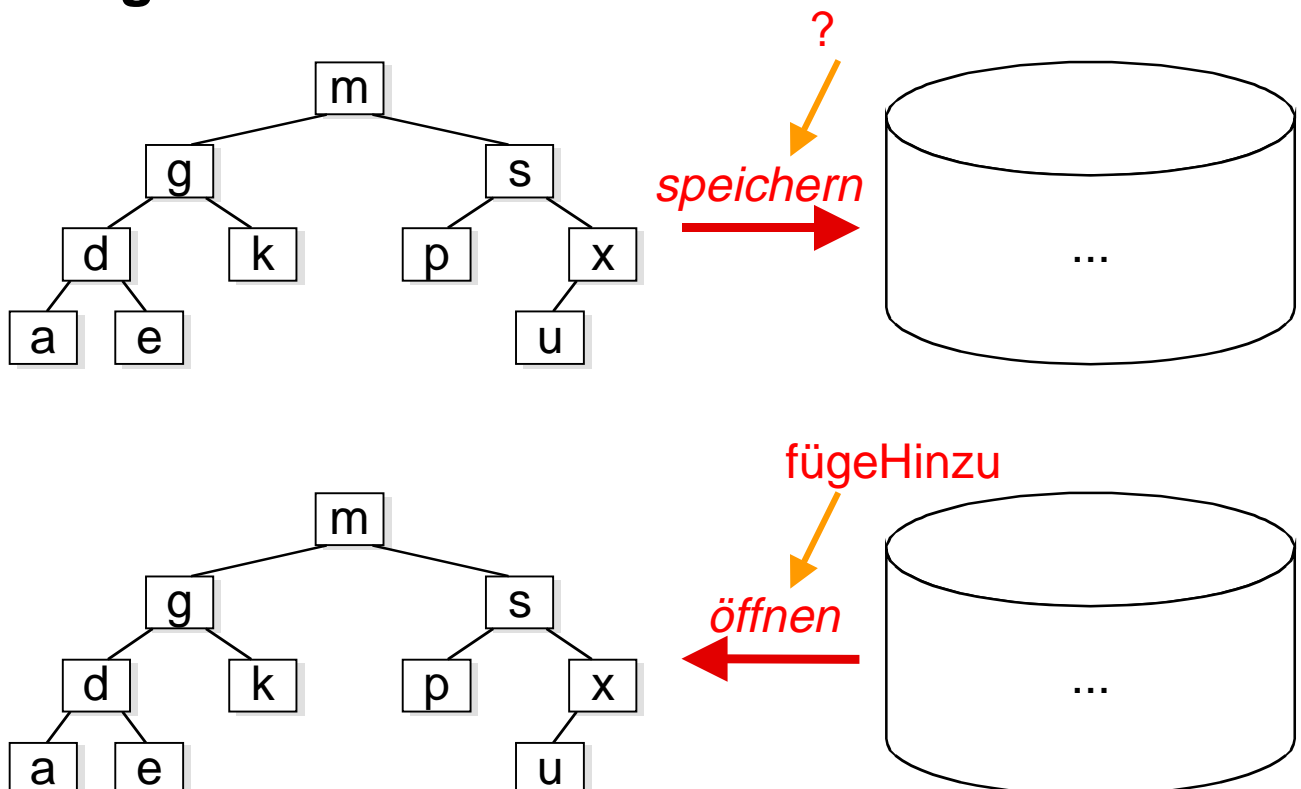
.....

# speichern

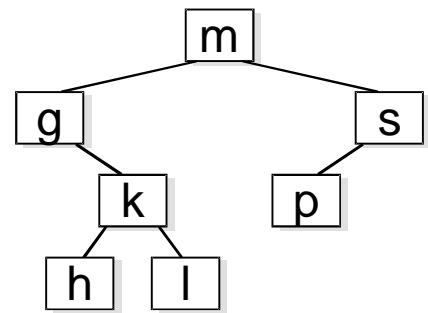
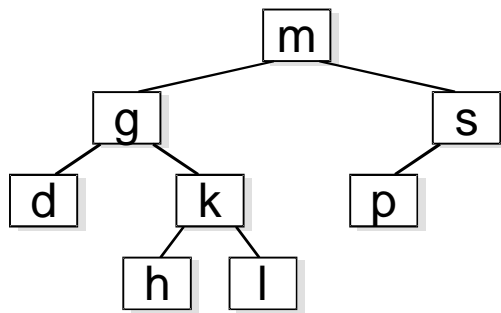
## Problem



## Lösung



# Liste deutsch-englisch Inordnungs-Traversierung



## Schnittstelle

- Collection-Objekte: For Each...Next-Schleife  
*Beispiel*

```
For Each Tab1 In Worksheets
```

```
...
```

```
Next Tab1
```

- CWörterbuch-Objekte: *keine* For Each...Next-Schleife möglich  
statt dessen zwei Funktionen:

**OErster()**: gibt den ersten Eintrag nach dem Inordnungs-Verfahren zurück

**ONächster(Eintrag)**: gibt den Ordnungsnachfolger von Eintrag nach dem Inordnungs-Verfahren zurück oder Nothing, falls kein Ordnungsnachfolger existiert

### Beispiel

```
Dim Knoten As cEintrag
```

```
Set Knoten = BuchDeEn.OErster()
```

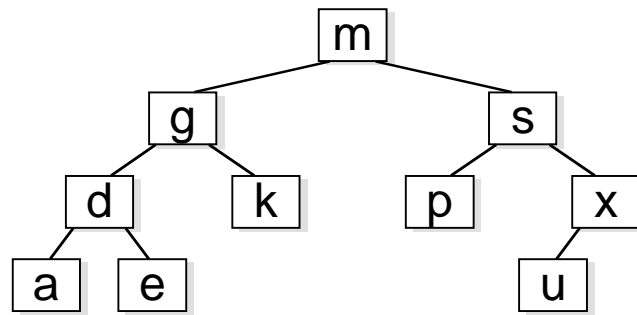
```
Do While Not Knoten Is Nothing
```

```
...
```

```
Set Knoten = BuchDeEn.ONächster(Knoten)
```

```
Loop
```

# speichern (Breitentraversierung)



## Schnittstelle

(als Ersatz für For Each...Next-Schleife)

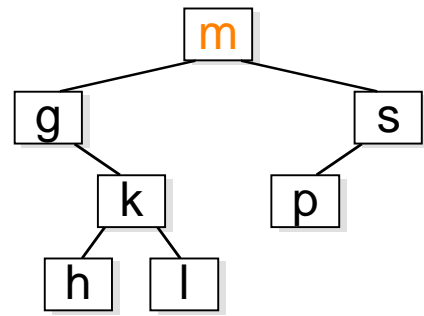
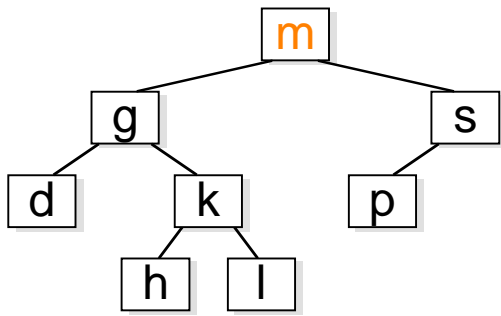
**NErster():** gibt den ersten Eintrag gemäss der Breiten-traversierung zurück

**NNächster(Eintrag):** gibt den Nachbar von Eintrag gemäss der Breitentraversierung zurück oder Nothing, falls kein Nachbar existiert

## Aufruf

```
Dim Knoten As cEintrag
Set Knoten = NErster()
Do While Not Knoten Is Nothing
    ...
    Set Knoten = NNächster(Knoten)
Loop
```

# Inordnungs-Traversierung



## Entwurf OErster()

### Beispiele

- OErsterRek('k')
- OErsterRek('g')
- OErsterRek('s')
- OErsterRek('d')
- OErsterRek('m')

### Entwurfscod

#### OErsterRek(**Baum**)

'Abbruchbedingungen

FALLS **Baum** leer ist  
gib Nothing zurück

SONST FALLS **Baum** KEINEN **linken** Teilbaum hat  
gib **Baum** zurück

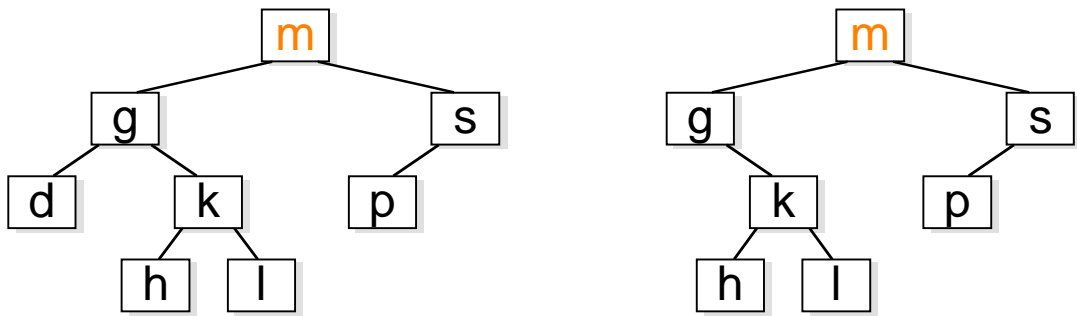
'Reduktionen

SONST  
gib ersten Eintrag von **linkem** Teilbaum von **Baum** zurück

### Test

⇒ Testen Sie den Algorithmus anhand der Beispiele

# Inordnungs-Traversierung



## Implementation OErster()

```
Public Function OErster() As cEintrag
    Set OErster = OErsterRek(Wurzel)
End Function
```

```
Private Function OErsterRek(Baum As cEintrag) _
    As cEintrag
```

```
    'Abbruchbedingungen:
```

```
If Baum Is Nothing Then
    Set OErsterRek = Nothing
```

```
ElseIf Baum.KindLi Is Nothing Then
    Set OErsterRek = Baum
```

```
    'Reduktionen
```

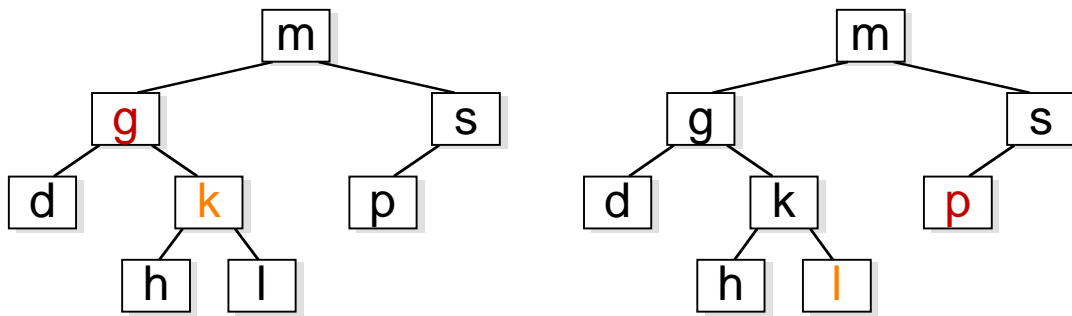
```
Else
```

```
    Set OErsterRek = OErsterRek(Baum.KindLi)
```

```
End If
```

```
End Function
```

# Inordnungs-Traversierung



## Entwurf ONächster()

### Beispiele

- ONächster('k')
- ONächster('g')
- ONächster('l')
- ONächster('p')
- ONächster('h')

### Entwurfscod

#### ONächster(Knoten)

FALLS Knoten einen rechten Teilbaum hat  
gib ersten Eintrag im rechten Teilbaum von Knoten zurück

SONST

beginne bei Knoten

WIEDERHOLE BIS aktueller Begriff > Begriff im Knoten

klettere einen Knoten hoch

FALLS neuer aktueller Knoten leer ist

gib Nothing zurück

brich ab

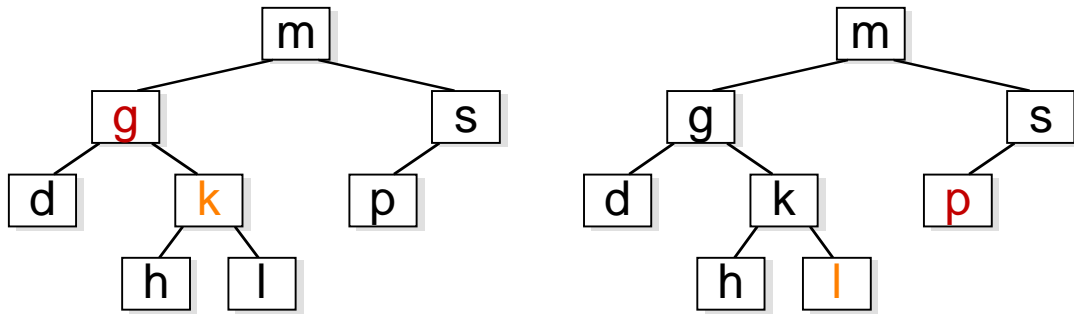
gib aktuellen Knoten zurück

### Test

Testen Sie den Algorithmus anhand der Beispiele



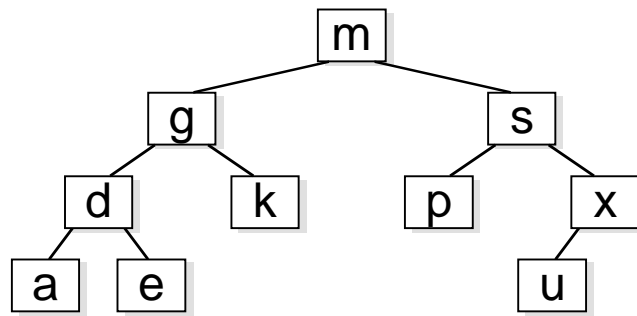
# Inordnungs-Traversierung



## Implementation ONächster()

```
Public Function ONächster(Knoten As cEintrag) _  
    As cEintrag  
  
    Dim Läufer As cEintrag  
    If Not Knoten.KindRe Is Nothing Then  
        Set ONächster = OErsterRek(Knoten.KindRe)  
    Else  
        Set Läufer = Knoten  
        Do Until Läufer.Begriff > Knoten.Begriff  
            Set Läufer = Läufer.Vater  
            If Läufer Is Nothing Then  
                Exit Do  
            End If  
        Loop  
        Set ONächster = Läufer  
    End If  
End Function
```

# Breitentraversierung



## Entwurf NErster()

### NErster()

FALLS Wurzel nicht leer ist  
gib Wurzel zurück

SONST  
gib Nothing zurück

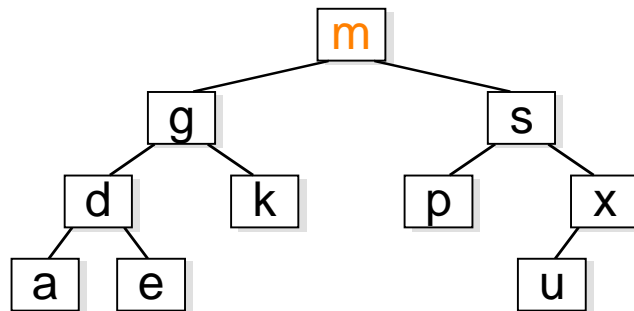
## Implementation NErster()

```
Public Function NErster() As cEintrag  
    Set NErster = Wurzel  
End Function
```

# Breitentraversion

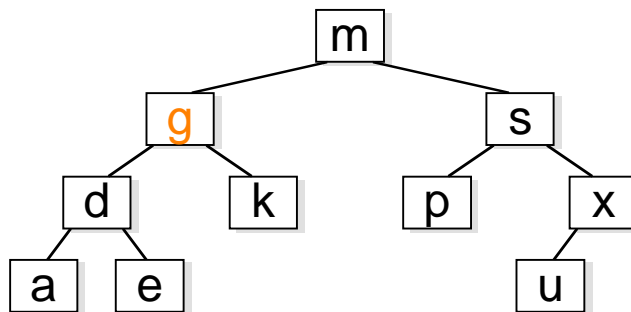
## zum Entwurf von NNächster()

1.



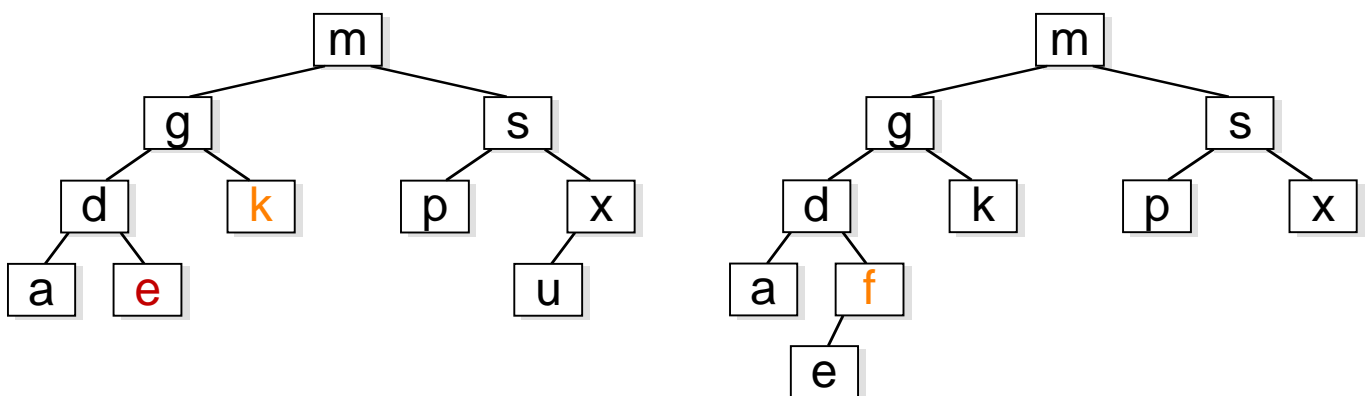
Der Nachbar der Wurzel ist ihr linkes bzw. ihr rechtes Kind. Hat die Wurzel keine Kinder, so hat sie auch keinen Nachbarn.

2.



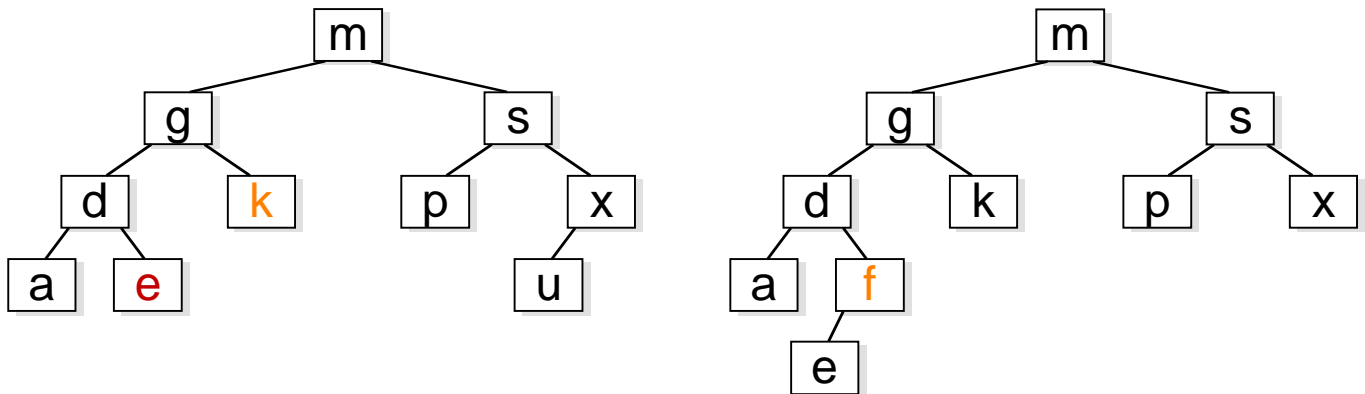
Der Nachbar eines linken Kindes mit einem rechten Bruder ist der rechte Bruder.

3.



Der Nachbar eines Kindes ohne rechten Bruder ist das linke bzw. das rechte Kind des nächsten (nicht kinderlosen) Nachbarn des Vaters.

# Breitenterversierung



## Entwurf NNächster()

### NNächster(Knoten)

FALLS Knoten = Wurzel

FALLS Knoten einen linken Nachfolger hat  
gib linken Nachfolger von Knoten zurück

SONST FALLS Knoten einen rechten Nachfolger hat  
gib rechten Nachfolger von Knoten zurück

SONST  
gib Nothing zurück

SONST  
FALLS Knoten linker Nachfolger seines Vaters ist  
UND Vater von Knoten einen rechten Nachfolger hat  
gib rechten Nachfolger von Vater von Knoten zurück

SONST  
Durchlaufe die Nachbarn<sup>(1)</sup> des Vaters von Knoten  
FALLS Nachbar einen linken Nachfolger hat  
gib linken Nachfolger des Nachbarn zurück  
FALLS Nachbar einen rechten Nachfolger hat  
gib rechten Nachfolger des Nachbarn zurück  
FALLS Nachbar = Knoten  
gib Nothing zurück  
brich ab

<sup>(1)</sup> Nachbar im Sinne der Breitenterversierung ⇒ Rekursion

## Implementation NNächster()

```
Public Function NNächster(Knoten As cEintrag) _
    As cEintrag
    Dim Läufer As cEintrag
    If Knoten Is Wurzel Then
        If Not Knoten.KindLi Is Nothing Then
            Set NNächster = Knoten.KindLi
        ElseIf Not Knoten.KindRe Is Nothing Then
            Set NNächster = Knoten.KindRe
        Else
            Set NNächster = Nothing
        End If
    Else
        If Knoten Is Knoten.Vater.KindLi _
        And Not Knoten.Vater.KindRe Is Nothing Then
            Set NNächster = Knoten.Vater.KindRe
        Else
            Set Läufer = Knoten.Vater
            Do While Not Läufer Is Knoten
                Set Läufer = NNächster(Läufer)
                If Not Läufer.KindLi Is Nothing Then
                    Set NNächster = Läufer.KindLi
                    Exit Do
                ElseIf Not Läufer.KindRe Is Nothing Then
                    Set NNächster = Läufer.KindRe
                    Exit Do
                End If
            Loop
        End If
    End If
End Function
```