

# Zusammenfassung

## Objektvariablen und Verweise

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

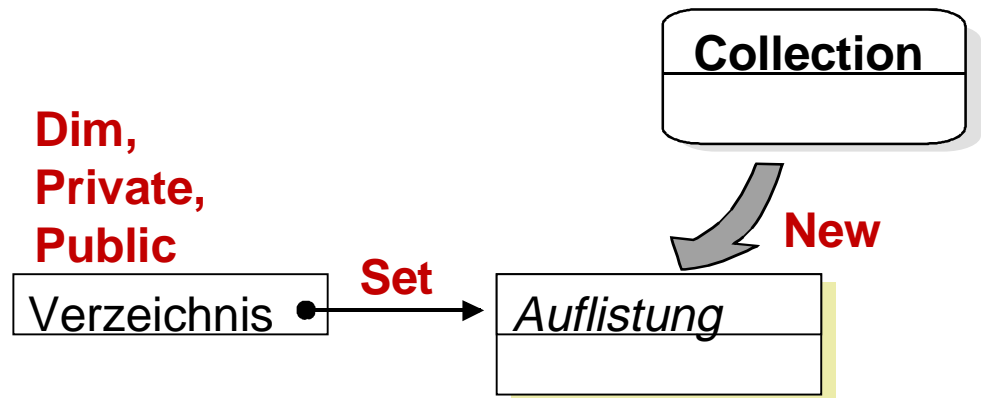
Ein **Verweis** entspricht der Anfangsadresse eines Speicherbereiches, in dem sich ein Objekt befindet.

- ⇒ 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

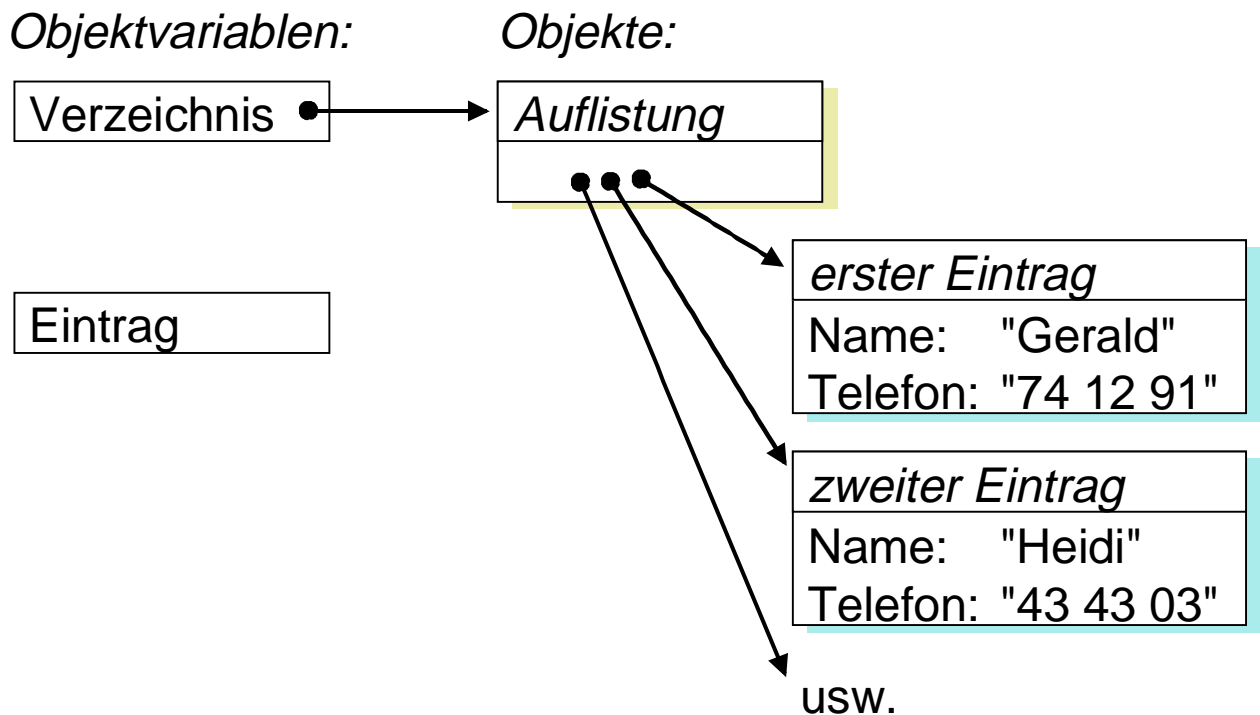


- 1) Objektvariable vereinbaren
- 2) Objekt erstellen und der Variablen zuweisen
- 3) über Variable auf Methoden und Eigenschaften zugreifen

## Beispiel

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

# Objektmodell des Telefonverzeichnisses



## 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

```
Sub Test()  
    Dim Temp As cEintrag  
    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.

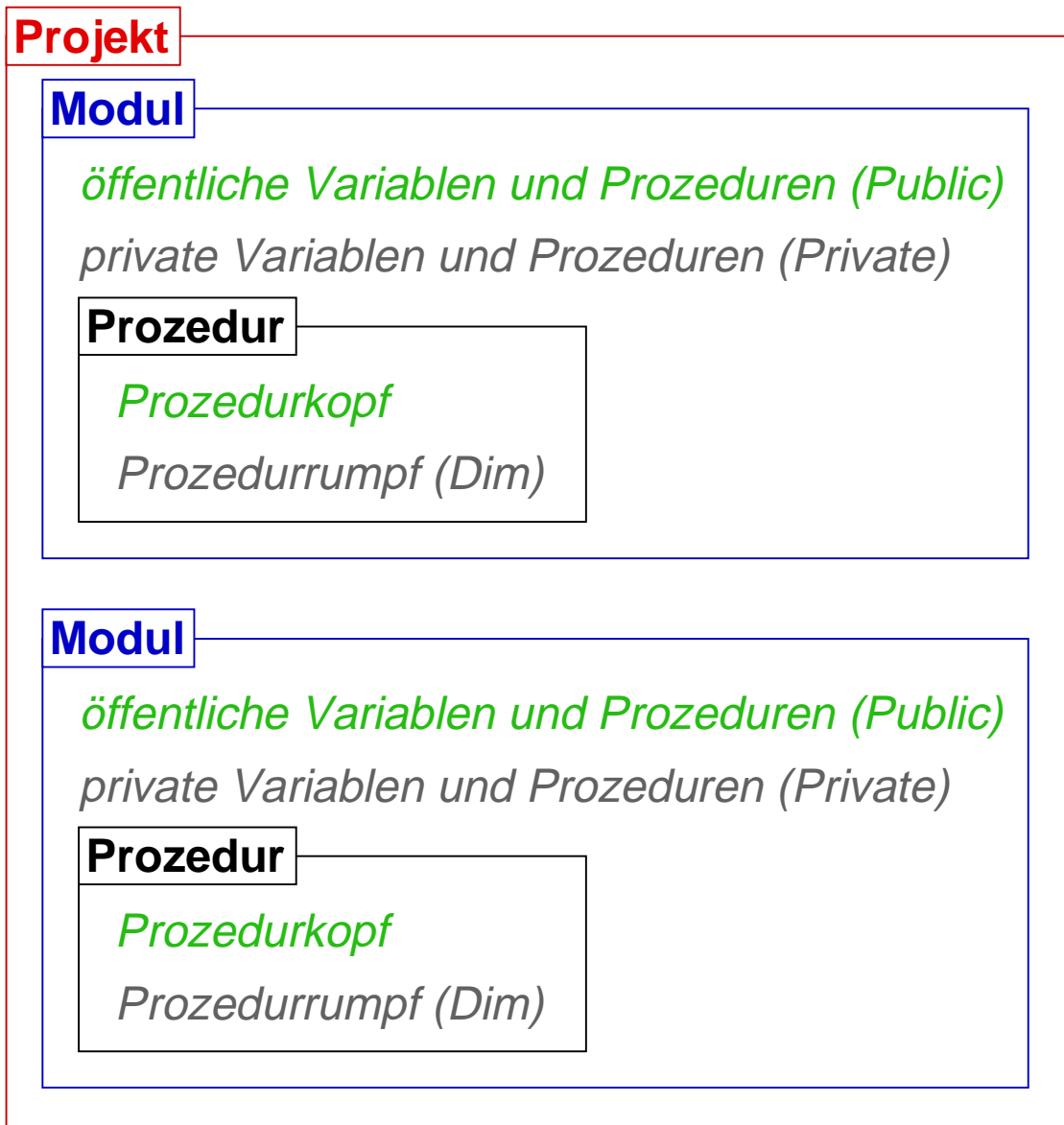
## Vergleich zwischen Collection und cEintrag

	<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. 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
```

## 2. 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

---

# 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:*

Eintrag1

Eintrag2

Temp



# 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
```

# 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

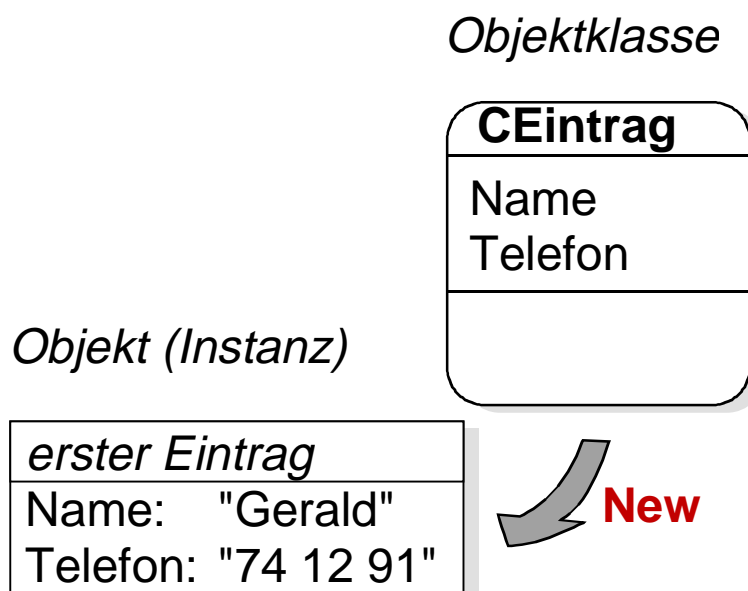
- ⇒ wichtige von unwichtigen Aspekten trennen
- ⇒ beschreiben von Eigenschaften und Methoden
- ⇒ modellieren der Wirklichkeit
- ⇒ kreieren abstrakter Datentypen

## 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

Klasse cTestA

```
Public Sub Zeige()  
    MsgBox "Ich bin ein Objekt vom Typ CTestA"  
End Sub
```

Klasse cTestB

```
Public Sub Zeige()  
    MsgBox "Ich bin ein Objekt vom Typ CTestB"  
End Sub
```

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
```

# 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

nur *erben* von Funktionalität beschränkt möglich

### Beispiel ([grüsseHöflich.xls](#))

Formular definieren

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



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

Formular verwenden

```
Private Dialog As frmGrüsse
Sub grüsseHöflich()
    If Dialog Is Nothing Then
        Set Dialog = New frmGrüsse
    End If

    Dialog.Gruss = "Guten Tag"
    Dialog.Show
End Sub
```