AEK 7
7. Access-Entwickler-Konferenz
Nürnberg, 25./26.9.2004

Praxiseinsatz von benutzerdefinierten Klassen in Microsoft® Access

Grundlagen – Beispiele – Tools

softconcept, Wien
www.softconcept.at

Paul Rohorzka
paul.rohorzka@softconcept.at

Ziele, Themen

- Sensibilisierung für wiederkehrende Aufgaben bei der Entwicklung unter Access
- Begriffe rund um „Klasse“/“Objekt“ abstecken
- Objekte korrekt einsetzen
- Benutzerdefinierte Klassen erstellen
- Fallstricke kennen
- Einsatzmöglichkeiten in Access kennenlernen
- Tools

Aufbau

- I. Motivation:
  Typische Szenarien bei der Access-Entwicklung
- II. Überblick:
  Programmieren mit Klassen in VB(A)
- III. Crash-Kurs:
  Entwickeln von Klassen in VB(A)
- IV. Konkret:
  Benutzerdefinierte Klassen im Praxiseinsatz
- V. Hilfreich:
  Tools zur Klassenentwicklung

I. Motivation:
Szenarien bei der Entwicklung mit Access

Anforderungen

Implementierungen

Probleme

Visionen

Problembereich 1: Konsistente GUI

- Wünsche
  - Konfigurierbarkeit von Symbolen, Farbschemata, Darstellungsarten (Benutzerbedürfnisse, CI, ...)
  - Formulare sollen sich letzte Position und Größe merken
  - Großer Bildschirm! $\rightarrow$ viel Platz im Formular!
- Implementierung ist möglich, aber
  - mühsam, aufwändig und fehleranfällig
  - erfordert häufiges Copy&Paste
  - Änderungen: Code jedes Formulars anpassen

Problembereich 1: Konsistente GUI (cont'd)

- Abhilfe
  - Implementierung der Funktionalität in Klasse
- Vorteile
  - Beliebig viele unabhängige Instanzen dieser Funktionalität
  - Ändern oder Nachrüsten von Features:
    Code muss oft nur in der Klasse geändert werden
- Demonstration
  - CFormAppearance
  - CFormPosSize

Problembereich 2: Standard-Reaktionen auf Ereignisse

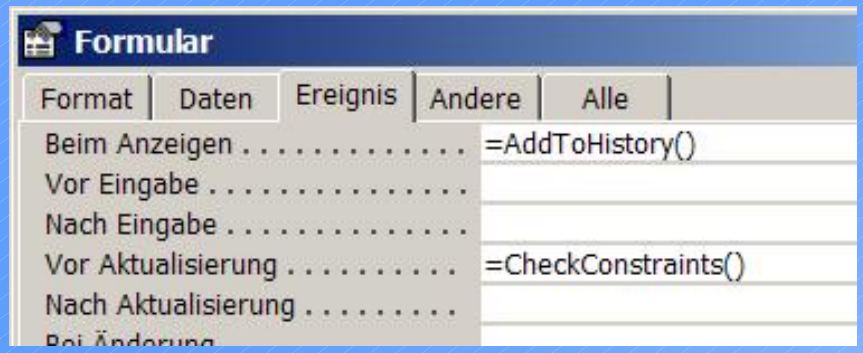
- Anforderungen
  - Schaltflächen mit Standardverhalten auf fast allen Formularen (Ok, Abbrechen, Übernehmen, Neu, ...)
  - Datensatzauswahl mit

```
Me.RecordSource = ... " WHERE ID=" & lngID
```

    (Performance)
  - Datensatz-History (~Web-Browser)
  - Klartextmeldungen für erforderliche Felder

Problembereich 2 (cont'd): Standard-Reaktionen auf Ereignisse

- Implementierung ist möglich, aber
  - ... (siehe oben)
  - Reaktion auf Ereignisse aus verschiedenen Formularen?



  - Ergänzungen, Anpassungen, (Fehler-)Suche?

Problembereich 2 (cont'd): Standard-Reaktionen auf Ereignisse

- Abhilfe:
  - Implementierung der Funktionalität in Klasse
  - Verwendung des Schlüsselwortes `WithEvents`
- Vorteile:
  - ... (siehe oben)
  - Klasse defin. auf welche Ereignisse wie reagiert wird
  - Reaktion auf weiteres Ereignis: Nur in Klasse!
- Demonstration
  - CFormHandling

Problembereich 3: Hantieren von Daten per Code

- Anforderung
  - Lesen und/oder schreiben von Daten abseits eines Formulars
- Implementierung ist möglich:

```
intAnzahl = DCount(...)
```

Pfui!

  - besser (?):

```
Dim rst As DAO.Recordset
Set rst = CurrentDb().OpenRecordset(...)
intAnzahl = rst!Anzahl
rst.Close
Set rst = Nothing
```

So viel Tipperei für eine Zahl?

Problembereich 3 (cont'd): Hantieren von Daten per Code

- Das wär schön:

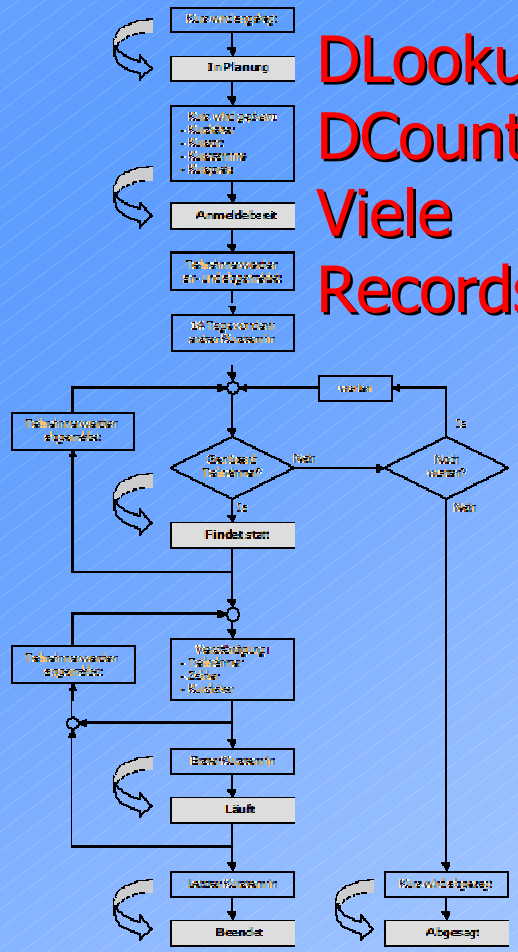
```
intAnzahl = Bestellungen.Count
```

- Oder komplexer:

```
With Customers(lngCustomerID)
    If .Saldo < 0 And .Saldo.Seit > 14 Then
        .Mahnstufe = .Mahnstufe + 1
        If .Mahnstufe > 3 Then
            Eintreiber.AddCustomer .Object
        End If
    End If
End With
```

Problembereich 3 (cont'd): Hantieren von Daten per Code

- Demonstration: Status eines Seminars bestimmen
  - Seminar freigegeben?
  - Zusage Seminarleiter?
  - Kursort fixiert?
  - Genügend Anmeldungen?
  - Seminar in Zukunft, läuft, beendet, abgesagt?
  - ...
  - → Infos aus 5 Tabellen



DLookup(), DCount()? Viele Recordsets?

Problembereich 3 (cont'd): Hantieren von Daten per Code

- Abhilfe
  - Implementieren von Kapselklassen um die Tabellen
- Vorteile
  - Objektbasierter Zugriff auf die Daten
  - Unterstützung durch Intellisense
  - Code liest sich beinahe wie Prosa
- Nachteile
  - Implementierung der Klassen ist nicht trivial
  - Wer implementiert die Klassen?

II. Überblick:
Programmieren mit Klassen in VB(A)

Elemente

Auflistungen

Ereignisse

Schnittstellen

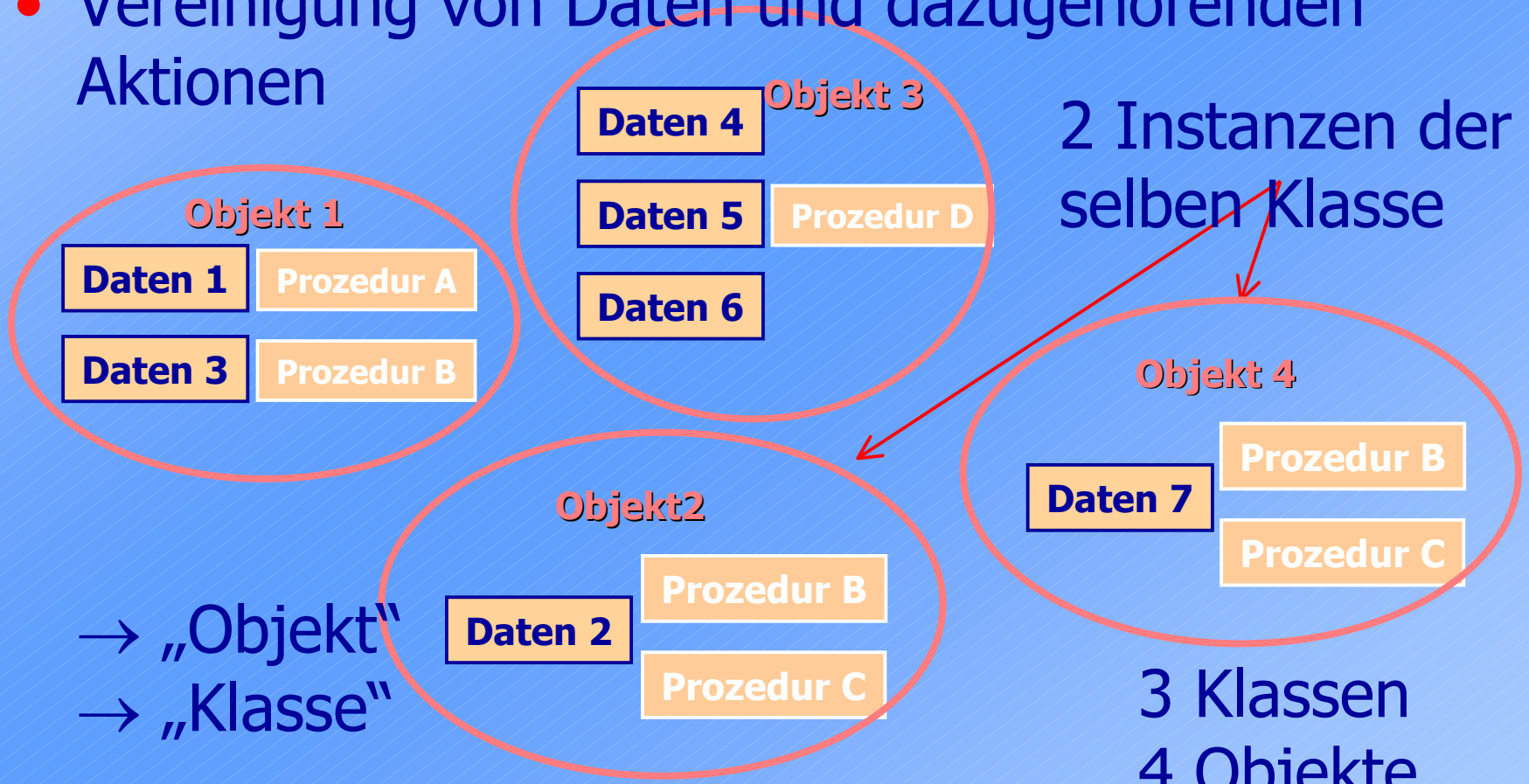
Prozeduraler Ansatz

- Trennung von Daten und Aktionen

- Problem
  - Zuordnung Daten/Prozeduren
  - Verwaltung der Daten

Objektbasierter Ansatz

- Vereinigung von Daten und dazugehörenden Aktionen

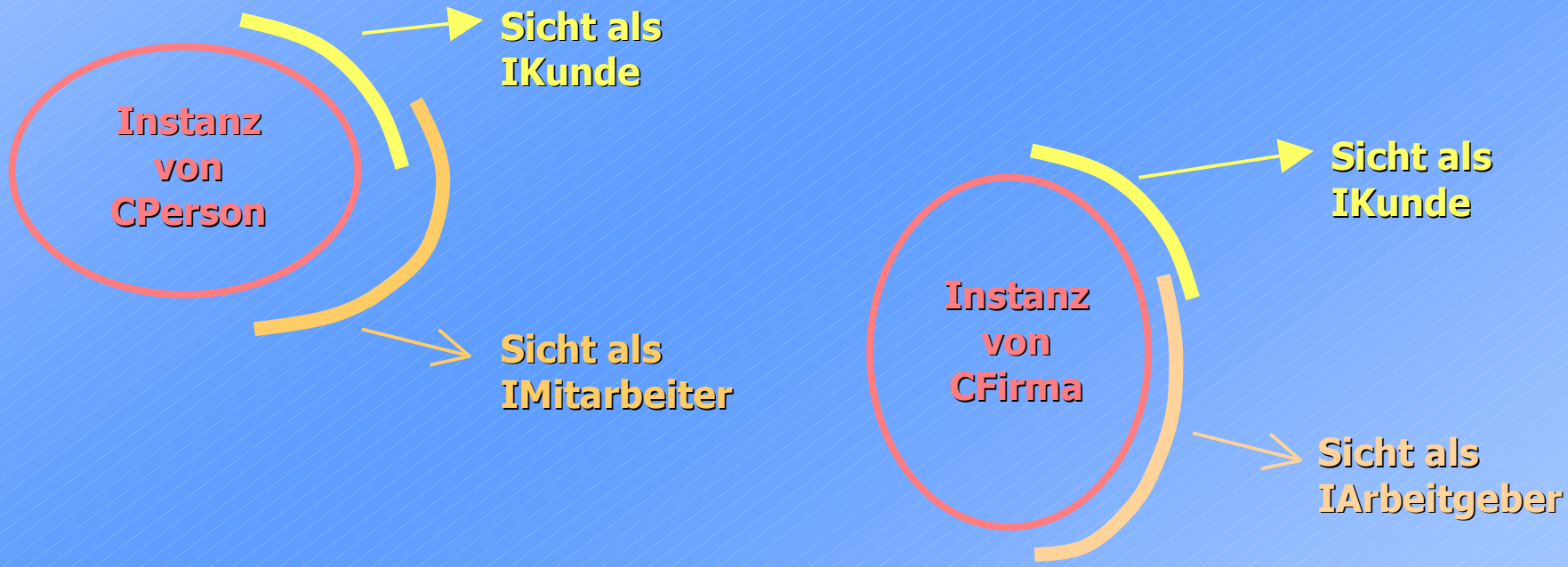


Objekte (Fortsetzung 2)

- Elemente eines Objekts:
  - Eigenschaften (=Daten)
  - Methoden (=Aktionen)
  - Ereignisse

Schnittstellen (Interfaces)

- Spezifische „Sicht“ auf ein Objekt



- „CPerson implementiert IKunde“

Entwickeln gegen eine Schnittstelle

- Code, der mit einem Objekt hantiert, erwartet nicht ein Objekt einer bestimmten Klasse, sondern ein Objekt, dessen Klasse eine bestimmte Schnittstelle implementiert.
- Beispiel von vorhin:
  - Code zum Abwickeln einer Bestellung erwartet nicht ein CPerson-Objekt, sondern IKunde.
  - $\rightarrow$ Der Code funktioniert ohne Änderung auch mit einem CFirma-Objekt

Wert vs. Referenztypen

- Werttyp
  - Variablenbezeichner repräsentiert den Wert
  - Eingebaute VB-Datentypen, Enum, UDT
  - Zuweisung: `[Let] meinWerttyp = derWert`
- Referenztyp
  - Variablenbezeichner repräsentiert einen Verweis auf das Objekt
  - Mehrere Verweise auf das selbe Objekt möglich
  - Alle Klasseninstanzen
  - Zuweisung: `Set meineReferenz = dasObjekt`

Lebensdauer eines Objekts

- Objekt (Klasseninstanz) erstellen: `New`

```
Set frmKundenA = New Form_frmKunden
```

  - mehrfach möglich

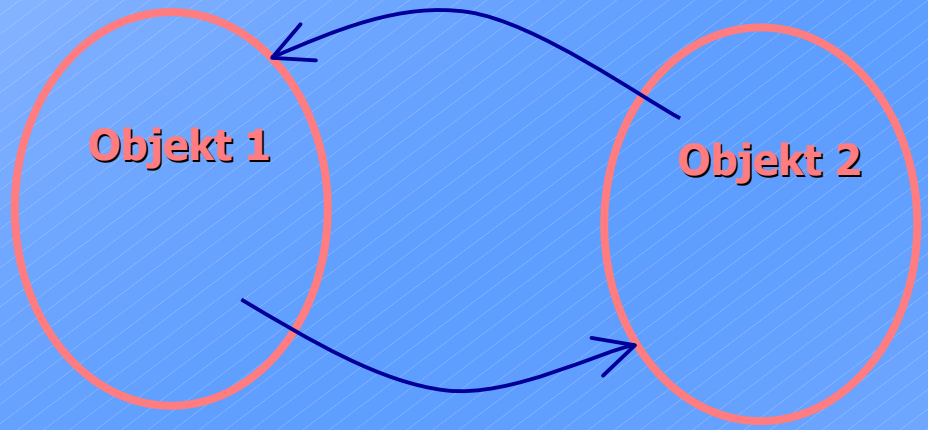
```
Set frmKundenB = New Form_frmKunden
```

- Objekt löschen
  - Nicht direkt möglich
  - Letzter Verweis gelöscht: Objekt wird gelöscht
- Objekt zählt Verweise auf sich
- Verweis löschen

```
Set frmKundenA = Nothing
```

Zirkuläre Referenzen

- Objekte verweisen wechselseitig aufeinander



- Problem: Verweisanzahl wird nie 0
- Lösung: Aufbrechen des Kreises

Zugriff auf Objekt-Elemente

- Eigenschaften

```
<Objekt>.<Eigenschaft>
txtNachname.FontName = "Tahoma"
```

- Methoden

```
<Objekt>.<Methode>
lstMeals.AddItem "Wiener Schnitzel"
```

- Ereignisse abfangen mit Prozedur

```
<Objekt>_<Ereignis>([<Parameterliste>])
Private Sub cmdClose_Click()
```

Standard-Elemente

- Jede Klasse kann ein Standard-Element haben (Eigenschaft oder Methode)
- Nennung kann unterbleiben:

```
lblInfo.Caption = "Alle belegt"
```

  - ist äuquivalent zu:

```
lblInfo = "Alle belegt"
```

- Hingegen

```
Set lblMeiner = lblUeberschrift
```

meint das Objekt selbst!

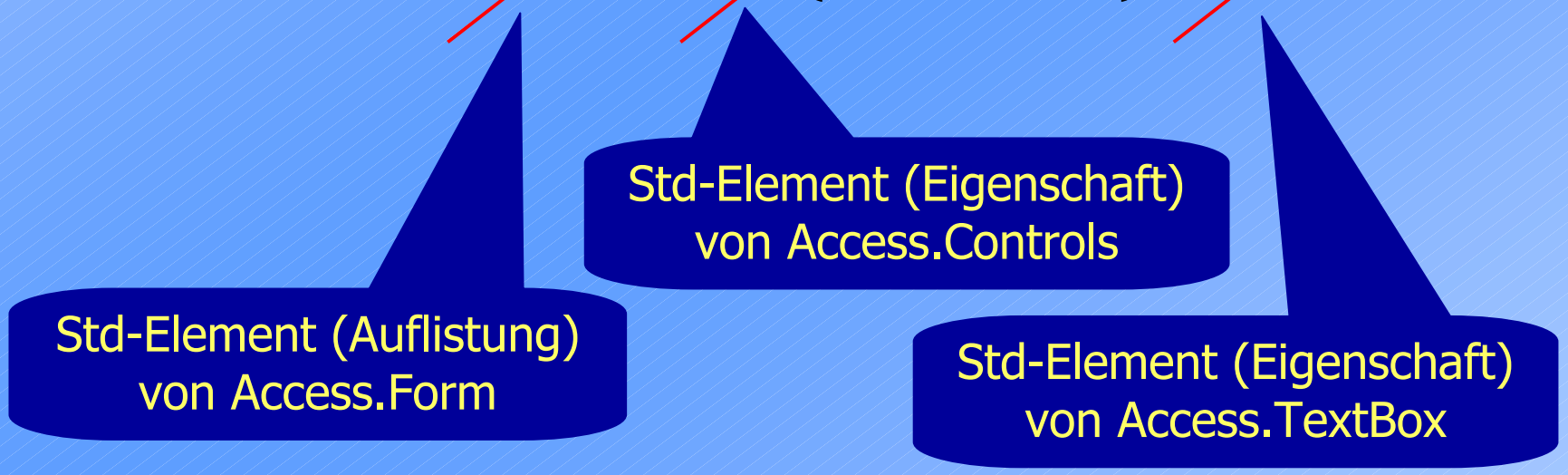
Standard-Elemente in Formular

- In Formular-Code:

```
Me!Nachname = Null
```

- Kurzschreibweise für

```
Me.Controls.Item("Nachname").Value = Null
```



Schlüsselwort Me

- Referenz auf das Objekt, das das aktuelle GUI-Formular kapselt (entsprechend bei Berichten)
- Datentyp ist Access.Form bzw. Access.Report
- Instanziierung übernimmt Access
- Referenz kann als Parameter übergeben werden:

```
' --- In Formular:
Call ShowTime(Me)

' --- In Standard-Modul:
Public Sub ShowTime(ByRef frm As Access.Form)
   frm.Caption = Now()
End Sub
```

Access-Objekte

- Geläufiger Code

```
Private Sub cmdClose_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

- Drei Objekte beteiligt:
  - cmdClose (löst das Ereignis Click aus)
  - DoCmd (zum Aufruf von GUI-Befehlen)
  - Me (repräsentiert das Formular)

III. Crash-Kurs:
Entwickeln von Klassen in VB(A)

Elemente

Auflistungen

Ereignisse

Schnittstellen

Klasse erstellen

- Klassenmodul einfügen
- Name des Klassenmoduls wird Name der Klasse
- Häufige Namenspräfixe:

  `CKunde`

  `clsKunde`

  `Kunde` (kein Präfix: gut zu lesen aber Gefahr der Verwirrung)

Eigenschaften erstellen 1

- Öffentliche Modulvariable

```
Public Nachname As String
```

  - Vorteil
    - Sehr einfach zu implementieren
  - Nachteil
    - Unkontrollierbarer Zugriff von außen
  - Am besten nie verwenden

Eigenschaften erstellen 2

- Zugriffs-Prozedur/Funktion:

```
Private mp_strNachname As String
...
Public Property Let Nachname(v As String)
   mp_strNachname = v
End Property
Public Property Get Nachname() As String
   Nachname = mp_strNachname
End Property
```

- Bei Referenztypen:
  `Property Set` statt `Property Let`

Eigenschaften erstellen 2 (cont'd)

- Vorteile:
  - Kontrollierbarer Zugriff
  - Information, dass Wert geändert wurde
  - Schreibgeschützte Eigenschaften (kein `Property Let`)
  - Berechnete (nicht gespeicherte) Eigenschaften:

```
Public Property Get Alter() As Date
    Alter = Date() - mp_dtmDOB
End Property
```

  - Parametrisierbare Eigenschaften
    - Bsp: Listenfeld: Column-Eigenschaft

Methoden erstellen

- Methoden sind (öffentliche) Prozeduren/Funktionen der Klasse

```
' --- In Klassenmodul CKunde:
Public Sub PrintOut()
    ...
End Sub
' --- In anderem Modul:
Dim objKunde As CKunde
Set objKunde = New CKunde
...
objKunde.PrintOut()
...
Set objKunde = Nothing
```

Ereignisse erstellen

- Deklaration

```
Public Event Changed(OldPrice As Currency)
```

- Auslösen

```
RaiseEvent Changed(strOldPrice)
```

- Abfangen des Ereignisses durch WithEvents

```
Dim WithEvents meineArtikel As CArtikel
...
Private Sub meinArtikel_Changed( _
   OldPrice As Currency)
```

- Rückmeldung an Objekt möglich
  - Parameterübergabe `ByRef`
  - Beispiel: Parameter Cancel in Open-Ereignis (Access.Form)

Konstruktion und Destruktion

- Klassenereignisse Initialize und Termine
  - Ereignisprozeduren
    `Class_Initialize` und `Class_Terminate`
  - Treten bei Beginn bzw. Ende der Lebensdauer auf
  - Für Initialisierungen bzw. Freigabe von Ressourcen
- `Class_Initialize`: Keine Parameter-übergabe bei Konstruktion möglich
  - Gefahr unvollständig initialisierter Objekte
  - Abhilfe: Factory-Funktion
    - Beispiel (DAO): OpenRecordset

Standard-Element erstellen

- ProzedurID 0 definiert Standard-Element
  - Problem: Einstellung des Attributs
  - Abhilfe: Text-Export-Trick
- Demonstration
  - Erstellen einer Standard-Eigenschaft

Schnittstelle erstellen

- Gewöhnliches Klassenmodul
  - Nur Deklarationen
  - Keine Implementierungen
  - Name oft `I...`
- Beispiel

```
Public Property Get Key() As String
End Property

Public Function Compare(o As Object) _
    As Integer
End Function
```

Schnittstelle implementieren

- Schlüsselwort `Implements`
- In Klassenmodul

```
Implements IComparable

Public Property Get IComparable_Key() _
   As String
   ... ' Implementierung hier
End Property

Public Function Get IComparable_Compare( _
   o As Object) As Integer
   ... ' Implementierung hier
End Function
```

Auflistungsklassen

- Fast immer vorhandene Elemente
  - Methode `Add`
  - Eigenschaft `Item` (Standard-Element)
  - Eigenschaft `Count`
  - Eigenschaft `NewEnum`
    (zur `For Each`-Unterstützung)
- Speicherung der Objekte intern am besten in `VBA.Collection`

For Each-Unterstützung

- Notwendig: Eigenschaft `NewEnum`
- ProzedurID -4 (als Attribut einstellen)
  - Problem: Einstellung des Attributs
  - Abhilfe: Text-Export-Trick
- Beispiel

```
Public Property Get NewEnum() As IUnknown
    Set NewEnum = m_coll.[_NewEnum]
End Property
```

- `NewEnum` wird 1x zu Beginn jeder `ForEach`-Schleife aufgerufen

IV. Konkret:
Benutzerdefinierte Klassen im Praxiseinsatz

Helfer

Compound-Controls

Datenhelfer

Datenkapsler

Helfer-Klassen (nicht Daten-gebunden) 1

- CNamedValues
  - Umwandlung einzelner benannter Werte in String

```
Dim nv As CNamedValues

Set nv = New CNamedValues
nv.Add "KundenID", Me.TeilnehmerID.Value
' Weitere: nv.Add <Name>, <Wert>
DoCmd.OpenForm gc_strFrmKursanmeldung, _
               OpenArgs := nv.AsString

Set nv = Nothing
```

Helfer-Klassen (nicht Daten-gebunden) 2

- CMoveAndSizeLabel
- CCoolButton
- CFormPosSize
- CKeyDownForwarder
- CDetailsHandler
- CFormAppearance

Helfer-Klassen (Daten-gebunden)

- CFormHandling
- CLogger
- COptionGlobal, COptionLocal, COptionUser

Datenkapsel-Klassen

- Beispiel: Bestimmung des Kursstatus
- Beteiligte Klassen:
  - CKurs
  - CKursort
  - CTermine
  - CTermin
  - CTeilnahmen

V. Hilfreich:
Tools zur Klassenentwicklung

Klassen-/Tabellengenerator

Fremdtools

- Objektbrowser (F2)
- VB6-Klassenassistent
- Class Builder Wizard (Ashish, Kreft)
- MZ-Tools
- XTG-DataModeler
- ...

Laufende Eigenentwicklungen

- scDbDevTools – u.a. Tabellengenerator
- sClassGen – Klassengenerator
- sCodeDocu – Dokumentation von VBA-Code
- scVBDebugTool – Hilfe beim Debugging

Persönliches Fazit

- ☺ Programmieren mit Objekten ist klasse!
- 😐 Klassen schreiben kann ziemlich mühsam sein!