assono

EntwicklerCamp 2015
Track 3, Session 3:
Wenn ich das früher gewusst hätte, hätte ich schon lange objekt-orientiert programmiert!

Gelsenkirchen, 3. März 2015

assono

Thomas Bahn

- Diplom-Mathematiker, Universität Hannover
- seit 1997 entwickle ich mit Java und relationalen Datenbanken
- seit 1999 mit Notes/Domino zu tun: Entwicklung, Administration, Beratung und Schulungen
- regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen zu IBM Lotus Notes/Domino und Autor für THE VIEW

tbahn@assono.de
www.assono.de/blog
04307/900-401

assono
IT-Consulting & Solutions

assono

Agenda

- Motivation
- Konzepte, Begriffe und Beispiele
- größeres Beispiel: Class EMail

assono

Motivation

- Du entwickelst in LotusScript?
- Dann hast du schon objekt-orientiert programmiert – jedes Mal wenn du NotesSession, NotesDatabase, NotesView, NotesDocument usw. verwendest.
- Aber sonst? Wer hat schon…
  - eigene Klassen geschrieben (Arme hoch!)
  - dabei bewusst Public und Private verwendet (Arme…)?
  - Unterklassen zu eigenen Klassen entwickelt (…)?
  - Unterklassen zu Produktklassen entwickelt (…)?

assono

Motivation (forts.)

1. Deine Arbeit wird einfacher, weil der Code besser lesbar und leichter wartbar ist.
2. Du musst weniger arbeiten, weil du mehr wieder verwenden kannst.
3. Du bekommst weniger graue Haare, weil Konflikte bei der Wiederverwendung viel seltener auftreten, und wenn du etwas wieder verwendest, kannst du es viel flexibler anpassen – ohne den alten Code zu ändern!
4. Du kannst die Komponenten (Klassen) getrennt entwickeln und testen und so die Entwicklung leicht auf mehrere Personen verteilen.

assono

Motivation (forts.)

5. Du brauchst viel weniger zu testen, weil du nur den neuen Code (Unterklassen) anschauen musst, weil der ursprüngliche Code (Oberklassen) unverändert bleibt.
6. Du lernst mit OOP etwas für deine Zukunft, weil es in modernen Programmiersprachen (Java, JavaScript) nicht mehr „ohne“ geht.
7. Du profitierst von der Arbeit anderer, denn es gibt viele fertige „Best-Practice-Rezepte“ für typische Probleme, die sogenannten Entwurfsmuster.

assono

Komplexität verstecken

- Wer weiß, wie NotesView genau implementiert ist?
- Wer muss das wissen, um die Klasse zu benutzen?
- Wir benutzen die öffentliche Schnittstelle zur Klasse, so wie sie in der Dokumentation steht, ohne zu wissen, wie das genau funktioniert – es nicht wichtig.
- Das ist der erste wichtige Prinzip von OOP:

  Die Komplexität der konkreten Implementierung ist vor dem Benutzer verborgen.

assono

Kapselung

- NotesView benutzt interne Datenstrukturen, wie den Verweis auf die Datenbank, Namen der Ansicht usw.
- Die Methoden, also die Prozeduren und Funktionen von NotesView, arbeiten auf diesen internen Daten.
- **Dieses Prinzip, dass Daten und Routinen, die zusammen gehören, auch zusammen definiert werden, heißt Kapselung.**
- Das führt zu mehr Übersicht und leichter wartbaren Code.

assono

Kapselung (forts.)

- Alternativen zu OOP nutzen normalerweise globale Variablen, auf die von vielen verschiedenen Code-Stellen zugegriffen wird.
- Probleme
  - Nachvollziehbarkeit Verhalten/Ausführungsreihenfolge?
  - Testen?
  - Fehlersuche?!?
  - Namenskonflikte im globalen Namensraum bei Kombination mehrerer „Code-Spenden“ wahrscheinlich

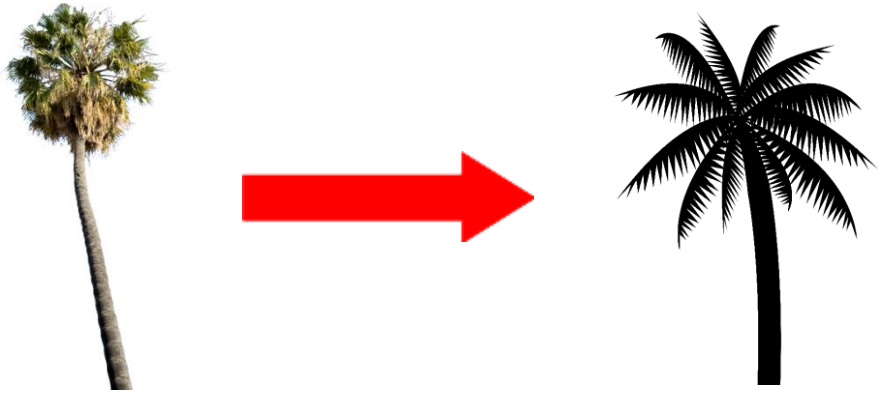
assono

Information-Hiding

- Man kann die internen Strukturen und Daten nur so weit direkt (von außen) sehen und ggf. verändern, wie das der Entwickler vorgesehen hat.
- **Das Prinzip, nur die für die Benutzung wichtigen Dinge nach außen für den Benutzer sichtbar zu machen und die internen Dinge zu verstecken, heißt Information-Hiding.**
- So kann die interne Implementierung nachträglich frei verändert werden, ohne das der Benutzer davon betroffen ist oder seinen Code anpassen müsste.

Klassen

- Was sind Klassen überhaupt?
  - Abstraktionen von der Realität
  - vereinfachtes Modell
    (für den eigenen Zweck Unwichtiges weg lassen)
  - Klassen beschreiben, welche Eigenschaften und Fähigkeiten etwas haben kann



assono

Klassen (forts.)

- (nutzloses) Beispiel: Autos
  - Farbe, Gewicht, Fahrgestellnummer, Geschwindigkeit, …
  - beschleunigen, bremsen, hupen, …
- Ein bestimmtes Auto hat dann in jedem Moment für jede Eigenschaft einen bestimmten „Wert“:
  - schwarz, 2 t, X96868FG4585, 98 km/h, …

assono

Klassen (forts.)

- weiteres Beispiel: Mitarbeiter
  - Personalnummer, Abteilung, Position, Durchwahl, Raum, Gehalt, …
  - befördere(neue Position, neues Gehalt)
  - zieheUm(neuer Raum, neue Durchwahl)
  - versetze(neue Abteilung)
- Ein Mitarbeiter hat zu einem Zeitpunkt:

Personalnummer: 4711	Abteilung: Marketing
Position: Abteilungsleiter	Durchwahl: -234
Raum: E334	Gehalt: 50.000 €

assono

Objekte/Instanzen

- Ein „konkreter“ Mitarbeiter ist ein **Objekt** oder eine **Instanz** der Klasse Mitarbeiter.
- Es gibt Objekte von realen Dingen (Autos, Mitarbeiter), aber auch von abstrakten (Bankkonto, Notes-Ansicht).
- „Think of an object as a fancy variable; it stores data, but you can „make requests“ to that object, asking it to do operations on itself.“
  Bruce Eckel, Thinking in Java

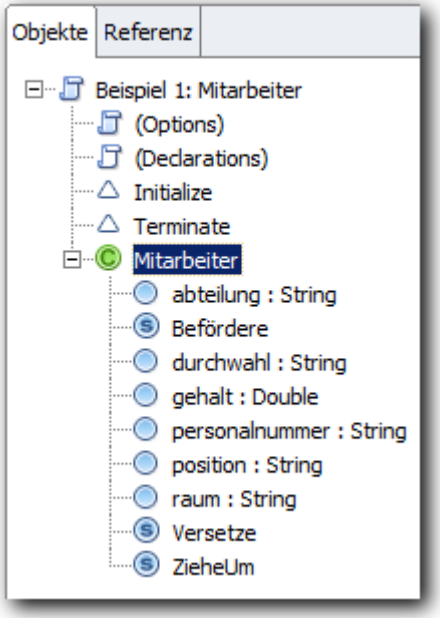
assono

Methoden und Attribute

- Routinen (Prozeduren und Funktionen) in Klassen nennt man **Methoden**,
  z. B. GetView in NotesDatabase
- Eigenschaften/Variablen in Klassen heißen **Attribute**,
  z. B. Title in NotesDatabase

Klasse und Objekt in LotusScript

- Noch einmal zum Mitarbeiter…
  - siehe Script-Bibliothek „Beispiel 01 - Mitarbeiter“ und Agent „Beispiel 01“
  - Klassen werden im Abschnitt (Declarations) definiert
  - im DDE ab Version 8.5 erscheinen sie als Einheit im Navigator:



assono

Beispiel 01: Mitarbeiter – Definition der Klasse

```
Class Mitarbeiter
    Public personalnummer As String
    Public abteilung As String
    Public position As String
    Public durchwahl As String
    Public raum As String
    Public gehalt As Double

[...]
```

assono

Beispiel 01: Mitarbeiter – Definition der Klasse (forts.)

```
[…]

    Public Sub Befördere(neuePosition As String, neuesGehalt As Double)
        position = neuePosition
        gehalt = neuesGehalt
    End Sub

    Public Sub ZieheUm(neuerRaum As String, neueDurchwahl As Double)
        neuerRaum = raum
        durchwahl = neueDurchwahl
    End Sub

    Public Sub Versetze(neueAbteilung As String)
        abteilung = neueAbteilung
    End Sub
End Class
```

assono

Neue Objekte/Instanzen erstellen

- Definition von Objekt-Variablen

Dim variable As Klasse

- notfalls funktionieren auch Variablen mit Datentyp Variant, aber Compiler kann dann nicht prüfen (Fehler dann zur Laufzeit!)
- Zuweisungen von Objekten immer mit **Set**

Set variable = neuerWert

- Neues Objekt aus Klasse erstellen mit **New**

Set variable = **New** Klasse

assono

Einschub: Namenskonventionen

- Namenskonventionen (die ich verwende):
  - lokale Variablen und Parameter beginnen klein:
    ma
  - Klassennamen beginnen groß:
    Mitarbeiter
  - Methoden (Prozeduren und Funktionen in Klassen) beginnen groß und enthalten ein Verb:
    GetView in NotesDatabase
  - Attribute (Variablen in Klassen) beginnen klein:
    gehalt in Mitarbeiter
  - zusammengesetzte Namen im „Camel Case“:
    NotesView

assono

Beispiel 01: Mitarbeiter – neues Objekt

```
Dim ma As Mitarbeiter

Set ma = New Mitarbeiter
ma.personalnummer = "4711"
ma.abteilung = "Marketing"
ma.position = "Abteilungsleiter"
ma.durchwahl = "-234"
ma.raum = "E334"
ma.gehalt = 50000

Print "Personalnummer", ma.personalnummer

Call ma.Versetze("Vertrieb")
Call ma.Befördere("Bereichsleiter", 60000)
```

assono

Konstruktor – eine ganz spezielle Methode

- Es gibt eine Methode, die automatisch einmal aufgerufen wird, wenn ein neues Objekt erstellt wird: der Konstruktor
- In LotusScript heißt er **New** und muss eine öffentliche Prozedur sein:

Public Sub New(optionale Parameterliste)

- typischerweise wird der Konstruktor genutzt, um wichtige Attribute zu initialisieren, insbesondere „Pflichtfelder“

assono

Beispiel 02: Mitarbeiter – Konstruktor

```
Class Mitarbeiter
    […]
    Public Sub New(neuePersonalnummer As String, _
    neueAbteilung As String, neuePosition As String, _
    neuesGehalt As Double)
        personalnummer = neuePersonalnummer
        abteilung = neueAbteilung
        position = neuePosition
        gehalt = neuesGehalt
    End Sub
    […]
End Class

Dim ma As Mitarbeiter
Set ma = New Mitarbeiter("4711", "Marketing", "Abteilungsleiter", 50000)
```

assono

Destruktor – gezielt aufräumen

- Vor dem Lebensende eines Objekts wird – sofern vorhanden – der Destruktor automatisch ausgeführt
- Bei lokalen Variablen: wenn der Block verlassen wird; sonst wenn das Objekt mit Delete gelöscht wird.
- In LotusScript heißt er **Delete** und muss eine öffentliche Prozedur sein:

Public Sub Delete()

- typischerweise wird der Konstruktor genutzt, um genutzte Ressourcen wieder frei zu geben, z. B. eine geöffnete Datei zu schließen

assono

Zugriffsmodifizierer: Public und Private

- Attribute und Methoden in Klassen sind entweder öffentlich (**Public**) oder privat (**Private**).
- Öffentliche Attribute/Methoden können von außen beliebig ausgelesen und verändert werden.
- Private Attribute/Methoden sind von außen nicht sichtbar und können nur von Methoden innerhalb der Klasse gelesen und verändert werden.
- Was ist, wenn man den Zugriff beschränken möchte, z. B. nur lesend, oder auf Veränderungen reagieren möchte, z. B. überprüfen oder protokollieren?

assono

Konvention: Getter und Setter

- Um den Zugriff beschränken zu können, **müssen** die Attribute und Methoden privat sein.
- Zugriff dann über öffentliche Methoden.
- Man nennt Methoden zum Auslesen von Attributen **Getter**, solche zum Setzen **Setter**.
- In Java gibt es die Konvention, den Namen von Gettern und Settern vom Attributnamen abzuleiten:
- Beispiel: Attribut gehalt ➔ GetGehalt und SetGehalt

assono

Beispiel 03: Mitarbeiter – Getter und Setter definieren

```
Class Mitarbeiter
    [...]
    Public Function GetPersonalnummer As String
        GetPersonalnummer = personalnummer
    End Function

    Private Sub SetPersonalnummer(neuePersonalnummer As String)
        personalnummer = neuePersonalnummer
    End Sub

    Public Function GetAbteilung As String
        GetAbteilung = abteilung
    End Function

    Public Sub SetAbteilung(neueAbteilung As String)
        abteilung = neueAbteilung
    End Sub
    [...]
End Class
```

assono

Beispiel 03: Mitarbeiter – Getter und Setter nutzen

```
Dim ma As Mitarbeiter

Set ma = New Mitarbeiter("4711", "Marketing", "Abteilungsleiter", 50000)
Call ma.SetDurchwahl("-234")
Call ma.SetRaum("E334")

Print "Personalnummer", ma.GetPersonalnummer


Call ma.SetPersonalnummer("4711")
```

➔ Compiler-Fehlermeldung:

Not a PUBLIC member: SETPERSONALNUMMER

assono

Alternative Getter- und Setter-Properties?

- Statt Prozeduren (Sub) und Funktionen kann man in LotusScript auch Get- und Set-Properties benutzen
- **Aber**: Get- und Set-Property müssen entweder beide öffentlich oder beide privat sein.
- Kompiler-Fehlermeldung: PROPERTY GET and SET must have the same storage class and visibility
- Deshalb verwende ich keine Properties, sondern Prozeduren und Funktionen.

assono

Beispiel 04: Mitarbeiter – Get-/Set-Properties

```
Class Mitarbeiter
    […]
    Public Property Get personalnummer As String
        personalnummer = personalnummer_
    End Property

    Public Property Set personalnummer As String
        personalnummer_ = personalnummer
    End Property

    Public Property Get abteilung As String
        abteilung = abteilung_
    End Property

    Public Property Set abteilung As String
        abteilung_ = abteilung
    End Property
    […]
End Class
```

assono

Beispiel 04: Mitarbeiter – Get-/Set-Properties nutzen

```
Dim ma As Mitarbeiter

Set ma = New Mitarbeiter("4711", "Marketing", "Abteilungsleiter", 50000)
ma.durchwahl = "-234"
ma.raum = "E334"

Print "Personalnummer", ma.personalnummer
```

assono

Unterklassen

- Es gibt nicht nur allgemein „Autos“, sondern auch Personenwagen, Lastwagen, Busse, …
- Diese Spezialisierungen haben jeweils andere, zusätzliche Eigenschaften, zum Beispiel:

Personen-Kfz	Lastkraftwagen	Bus
umklappbarer Rücksitz	Ladungskapazität	Passagierkapazität

- in OOP spricht man dann von **Unterklassen**
- Lastkraftwagen ist **Unterklasse** von Auto.
- Auto ist **Oberklasse** von Lastkraftwagen.

assono

Vererbung

- Ein Lastkraftwagen ist immer ein Auto, aber nicht jedes Auto ist ein Lastkraftwagen.
- Lastkraftwagen haben alle Eigenschaften (Attribute) und Fähigkeiten (Methoden) von Auto und ggf. weitere.
- Man sagt: Die (Unter-)Klasse Lastkraftwagen erbt von der (Ober-)Klasse Auto

assono

Vererbung (forts.)

- anderes Beispiel:
  NotesBenutzer sei eine Unterklasse von Mitarbeiter
- Schreibweise in LotusScript:

Class NotesBenutzer **As** Mitarbeiter

- Jeder NotesBenutzer ist also ein Mitarbeiter und hat erst einmal die gleichen Attribute und Methoden.
- Zusätzlich hat NotesBenutzer das Attribut NotesName und die Methode HoleMailDB
- Methoden von Unterklassen dürfen auch auf private Elemente der Oberklasse zugreifen.

assono

Konstruktoren/Destruktoren von Unterklassen

- Hat die Oberklasse einen Konstruktor (Sub New) **mit** Parametern, **muss** auch die Unterklasse einen Konstruktor haben.
- Wird ein Objekt erzeugt, wird erst der Konstruktor der Oberklasse(n) ausgeführt, dann der der Unterklasse.
- Bei Destruktoren genau anders herum: zuerst der Destruktor der Unterklasse, dann der bzw. die der Oberklasse.

assono

Konstruktoren von Unterklassen (forts.)

- In LotusScript kann man den Konstruktoren der Oberklasse(n) Parameter weitergeben.
- Beispiel für NotesBenutzer:

```
Public Sub New(neuePNr As String, neueAbteilung As String, _
    neuePosition As String, neuesGehalt As Double, _
    neuerNotesName As String), _
    Mitarbeiter(neuePNr, neueAbteilung, neuePosition, neuesGehalt)

    notesName = neuerNotesName
End Sub
```

- ruft den Konstruktor von Mitarbeiter mit den angegebenen Parametern auf

assono

Beispiel 05: Unterklasse NotesBenutzer definieren

```
Class NotesBenutzer As Mitarbeiter
    Private notesName As String

    Public Function GetNotesName As String
        GetNotesName = notesName
    End Function

    Private Sub SetNotesName(neuerNotesName As String)
        notesName = neuerNotesName
    End Sub

[...]
```

assono

Beispiel 05: Unterklasse NotesBenutzer definieren (forts.)

```
[…]

    Public Sub New(neuePNr As String, neueAbteilung As String, _
    neuePosition As String, neuesGehalt As Double, _
    neuerNotesName As String), _
    Mitarbeiter(neuePNr, neueAbteilung, neuePosition, neuesGehalt)
        notesName = neuerNotesName
    End Sub

    Public Function HoleMailDB As NotesDatabase
        Set HoleMailDB = Nothing
        '...
    End Function
End Class
```

assono

Beispiel 05: Unterklasse NotesBenutzer benutzen

```
Dim nn As NotesBenutzer

Set nn = New NotesBenutzer("4711", "Marketing", "Abteilungsleiter", _
                                                50000, "CN=Rainer Zufall/O=assono")
Call nn.SetDurchwahl("-234")
Call nn.SetRaum("E334")

Call nn.Versetze("Vertrieb")
Call nn.Befördere("Bereichsleiter", 60000)

Print "Personalnummer", nn.GetPersonalnummer
Print "Gehalt", nn.GetGehalt
Print "NotesName", nn.GetNotesName
```

assono

Objekte sind Werte…

- Objekte sind Werte und können…
  - in Variablen, Arrays, Listen und Attributen anderer Objekte gespeichert werden
  - an Prozeduren, Funktionen und Methoden als Parameter übergeben werden
  - von Funktionen zurückgegeben werden
- Objekte werden immer als Referenz übergeben
- Objekte dürfen auch in Variablen mit dem Typ einer Oberklasse gespeichert werden.
- Dann dürfen aber nur die Attribute und Methoden der Oberklasse verwendet werden.

assono

Beispiel 06: Objekte als Werte

```
(Declarations)
Dim alleBenutzer(0 To 1) As NotesBenutzer
Dim benutzerListe List As NotesBenutzer


Public Sub Initialize
    Set alleBenutzer(0) = New NotesBenutzer("4711", "Marketing", _
                          "Abteilungsleiter", 50000, "CN=Rainer Zufall/O=assono")
    Set alleBenutzer(1) = New NotesBenutzer("4712", "Marketing", _
                          "Mitarbeiter", 40000, "CN=Michael Berger/O=assono")

    ForAll benutzer In alleBenutzer
        Set benutzerListe(benutzer.GetPersonalnummer()) = benutzer
    End ForAll
End Sub
```

assono

Beispiel 06: Objekte als Werte (forts.)

```
Public Function SucheBenutzer(suchstring As String) As NotesBenutzer
    Dim gefunden As BooleanSet

    SucheBenutzer = Nothing
    ForAll benutzer In alleBenutzer
        gefunden = InStr(benutzer.GetPersonalnummer(), suchstring) > 0 Or _
        InStr(benutzer.GetAbteilung(), suchstring) > 0 Or _
        InStr(benutzer.GetPosition(), suchstring) > 0  Or _
        InStr(benutzer.GetDurchwahl(), suchstring) > 0  Or _
        InStr(benutzer.GetRaum(), suchstring) > 0  Or _
        InStr(benutzer.GetNotesName(), suchstring) > 0

        If gefunden Then
            Set SucheBenutzer = benutzer
            Exit Function
        End If
    End ForAll
End Function
```

assono

Beispiel 06: Objekte als Werte (forts.)

```
Dim nn As NotesBenutzer
Dim ma As Mitarbeiter

Set nn = benutzerListe("4711")
Print nn.GetNotesName

Set nn = SucheBenutzer("Michael")
Print nn.GetNotesName

Set ma = nn
Print ma.GetPersonalnummer
```

assono

Unterklassen definieren Methoden um (Polymorphismus)

- In einer Unterklasse wird eine Methode, die es auch in der Oberklasse gibt, einfach noch einmal definiert.
- Wird die Methode auf einem Objekt aufgerufen, sucht die LotusScript-Laufzeitumgebung von „unten nach oben“ nach einer Implementierung.

assono

Unterklassen definieren Methoden um (forts.)

- Die Methode der Unterklasse überdeckt dabei die der Oberklasse – man sagt, sie **überschreibt** diese.
- Sie – und nur sie – kann die überschriebene Methode der Oberklasse aufrufen. Schreibweise:

```
Class Unterklasse As Oberklasse
    Public Sub tueDas()
        […]
        Call Oberklasse..tueDas()
        […]
```

assono

Beispiel 07: Polymorphismus

```
Class Mitarbeiter

Public Function ToString As String
    ToString = "Mitarbeiter ["  & Chr$(10)
    ToString = ToString & " Personalnummer: " & personalnummer & Chr$(10)
    ToString = ToString & " Abteilung: " & abteilung & Chr$(10)
    ToString = ToString & " Position: " & position & Chr$(10)
    ToString = ToString & " Durchwahl: " & durchwahl & Chr$(10)
    ToString = ToString & " Raum: " & raum & Chr$(10)
    ToString = ToString & " Gehalt: " & gehalt & Chr$(10)
    ToString = ToString & "]"
End Function
```

assono

Beispiel 07: Polymorphismus (forts.)

```
Class NotesBenutzer As Mitarbeiter

Public Function ToString As String
    ToString = StrLeftBack(StrRight$(Mitarbeiter..ToString, " ["), "]")

    ToString = "NotesBenutzer ["  & Chr$(10) & ToString
    ToString = ToString & "     NotesName: " & notesName & Chr$(10)
    ToString = ToString & "]"
End Function
```

assono

Beispiel 07: Polymorphismus (forts.)

```
Dim nn As NotesBenutzer
Dim ma As Mitarbeiter

Set nn = benutzerListe("4711")
Print nn.ToString

Set nn = SucheBenutzer("Michael")
Print nn.ToString

Set ma = nn
Print ma.ToString
```

assono

Polymorphismus – Erkenntnis

- Es wird jedes Mal NotesBenutzer.ToString ausgeführt, auch wenn das NotesBenutzer-Objekt in einer Mitarbeiter-Variable gespeichert ist
- Man kann mit Unterklassen das „Verhalten“ ändern, indem man Methoden quasi ersetzt **ohne dass der ursprüngliche Code (Oberklasse) verändert wird**

assono

Me und TypeName

- Me verweist in Methoden auf das aktuelle Objekt
- TypeName(Objekt) liefert den Klassennamen
- TypeName(Me) gibt Klassennamen des aktuellen Objekts zurück (komplett groß geschrieben)
- ToString könnte auch so beginnen:

```
Public Function ToString As String
    ToString = TypeName(Me) & " [" & Chr$(10)
    […]
```

assono

Delegation

- Man kann keine Unterklassen von Produktklassen (NotesSession, -Database usw.) erstellen.
- Klassen können objektwertige Attribute haben, darunter auch Objekte von Produktklassen.
- Die zu erledigende Arbeit wird an das Objekt-Attribut weiter gegeben, also delegiert.
- Eine eigene Klasse kann also Produktklassen nutzen!

assono

Delegation vs. Vererbung

- Vererbung:
  - A **ist** ein(e) B
  - LKW ist ein Auto
  - zwischen Klassen
  - statisch/beim Kompilieren
- Delegation:
  - A **hat** ein(e) B
  - Auto hat ein Motor
  - zwischen Objekten
  - dynamisch/zur Laufzeit

assono

Beispiel 08: Delegation

```
Class NotesBenutzer As Mitarbeiter
    Private mailDB As NotesDatabase

    Private Sub SetMailDB(neueMailDB As NotesDatabase)
        Set mailDB = neueMailDB
    End Sub
```

assono

Beispiel 08: Delegation (forts.)

```
Public Function GetMailDB As NotesDatabase
    Dim domDirDB As NotesDatabase
    Dim personsColl As NotesDocumentCollection
    Dim personDoc As NotesDocument

    If mailDB Is Nothing Then
        Set domDirDB = New NotesDatabase("", "names.nsf")
        Set personsColl = domDirDB.Search(|Type = "Person" & | & _
                                |Fullname = "| & notesName & |"|, Nothing, 0)
        Set personDoc = personsColl.GetFirstDocument
        Set mailDB = New NotesDatabase(personDoc.MailServer(0), _
                                                    personDoc.MailFile(0))
    End If
    Set GetMailDB = mailDB
End Function
```

assono

Beispiel 08: Delegation (forts.)

```
Public Function GetMailDBGröße As Double
    GetMailDBGröße = GetMailDB().Size
End Function

Public Function GetMailDBGrößenWarnung As Double
    GetMailDBGrößenWarnung = GetMailDB().SizeWarning
End Function

Public Sub SetMailDBGrößenWarnung(neueGrößenWarnung As Double)
    GetMailDB().SizeWarning = neueGrößenWarnung
End Sub

Public Function GetMailDBGrößenQuote As Double
    GetMailDBGrößenQuote = GetMailDB().SizeQuota
End Function

Public Sub SetMailDBGrößenQuote(neueGrößenQuote As Double)
    GetMailDB().SizeQuota = neueGrößenQuote
End Sub
```

assono

Beispiel 08: Delegation (forts.)

```
Dim nn As NotesBenutzer

Set nn = benutzerListe("4711")
Print "NotesName: ", nn.ToString

Print "aktuelle Größe: ", Format(nn.GetMailDBGröße, "0,000")
Print "aktuelle Quota: ", Format(nn.GetMailDBGrößenQuote, "0,000")

' neue Quota = aktuelle Größe + 10 MB
Call nn.SetMailDBGrößenQuote(nn.GetMailDBGröße + 10485760)

Print "neue Quota: ", Format(nn.GetMailDBGrößenQuote, "0,000")
```

assono

Routinen als Parameter übergeben/zurückgeben

- In LotusScript können Routinen (Prozeduren und Funktionen) nicht als Parameter übergeben oder als Rückgabewert einer Funktion zurückgegeben werden.
- Aber: Man kann Objekte übergeben und zurückgeben.
- Lösung: Funktion als Methode in Klasse „stecken“, Objekt der Klasse erzeugen und das übergeben!
- Beispiel: verschiedene Sortieralgorithmen, von denen einer nach bestimmten Kriterien ausgewählt und verwendet wird
- Fachjargon: Strategie-Entwurfsmuster

assono

Trick: flexibler Parameter

- Objekt wird initialisiert aus Dokument oder über eine ID

```
Public Sub New(config As Variant)
    Dim configDoc As NotesDocument

    If LCase(TypeName(config)) = "notesdocument" Then
        Set configDoc = config
    Else
        Me.jobID = CStr(config)
        Set configDoc = GetConfigDoc("ExportJobsNachJobID", jobID)
    End If
```

assono

Wo verwende ich OOP?

- UI mit Model-View-Controller-Pattern (MVC)
  - zu jeder Maske eine Model-Klasse, die z. B. verantwortlich ist für Validierungen und
  - eine Controller-Klasse, die zwischen UIDocument und Model-Klasse vermittelt
- Import und Synchronisation
  - Klasse für Datensätze mit Lese-Methode (ReadFromDoc)
  - nicht NotesDocument, sondern DocUNID speichern
  - alle Datensätze in den Speicher einlesen (Objektliste)
  - IsElement(Liste) statt View-Lookup ➔ **viel** effizienter

assono

Wo verwende ich OOP? (forts.)

- Konfigurationen, Schlüsselwerte
  - Klasse für Konfiguration/Schlüsselwerte mit Lese-Methode
  - Einstellungen werden einmal eingelesen und können von überall in LotusScript leicht verwendet werden
- Relationen zwischen Masken
- Lösch-Schutz auf Datenbankebene (QueryDelete)

assono

Wo verwende ich OOP? (forts.)

- Fehlerbehandlung und –protokollierung
  - ErrorDetails, ErrorContext, CodePosition
  - ErrorHandler: IgnoreErrorHandler, LoggingHandler, EMailNotification, MessageboxHandler, WebAgentErrorHandler, WebFormErrorHandler
  - Logger: EMailLogger, DBLogger
  - Array von ErrorHandler-Objekten wird durchlaufen
    - zuständig? – dann ausführen
    - erledigt oder weiter mit dem nächsten?
    - ID, Priorität, aktiv/inaktiv, nur einmal ausführen

assono

Wo verwende ich OOP? (forts.)

- Hilfsklassen
  - RichTextHelper: vereinfacht Umgang mit RichText
  - Schnittstelle zu Word/Excel/PowerPoint
  - Zugriff auf das Domino Verzeichnis und Adressbücher
  - Erstellen von Kalendereinträgen und Aufgaben
  - E-Mail: super flexibles Werkzeug, E-Mails zu erzeugen
    - auch HTML/MIME-E-Mails
    - auch aus Vorlagen
    - auch mit Setzen des (sichtbaren) Absenders

assono

Class EMail – Alternative: Prozeduren

- Warum als Klasse?
- Vorher Prozeduren:
  - SendMail(empfänger, subject, body)
  - SendMailMehrere(empfänger, cc, bcc, subject, body)
  - SendMailDocLink(empfänger, subject, body, link, comment)
  - SendMailMehrereDocLink(…)
  - SendMailAbsender(empfänger, absender, subject, body)
  - SendMailAbsenderMehrere(…)
  - SendMailAbsenderMehrereDocLink(…)
  - …DirektVersenden…
  - …ZusätzlicheItems…

assono

Class EMail – Klasse ist die Lösung

- sehr viele Varianten, wenn man alle möglichen Kombinationen haben möchte – oder eine Funktion mit „unendlicher“ Parameterliste, wovon die meisten beim Aufruf leer bleiben
- mit Klasse: alles möglich, nichts muss

assono

Class EMail – Verwendungsbeispiele

- Objekt erstellen und gewünschte Eigenschaften setzen

```
Dim mail As EMail

Set mail = New EMail()
Call mail.SetSendTo("tbahn@assono.de")
Call mail.SetSubject("Erster Test")
Call mail.AppendText("Erster Test")
Call mail.Send
```

- mit „Call Chaining“

```
Set mail = New EMail()
Call mail.SetSendTo("tbahn@assono.de").SetSubject("Zweiter Test")._
                                      AppendText("Zweiter Test").Send
```

assono

Trick: Call Chaining

- Methode gibt Referenz auf das aktuelle Objekt zurück
- Aus

```
Public Sub SetSender(sender As String)
    Me.sender = sender
End Sub
```

- wird

```
Public Function SetSender(sender As String) As EMail
    Me.sender = sender
    Set SetSender = Me
End Function
```

- So kann der nächste Methodenaufruf für das EMail-Objekts direkt wieder angehängt werden

assono

Class EMail – Verwendungsbeispiele II

- Komplexe E-Mail erstellen und senden

```
Dim session As New NotesSession
Dim currentDB As NotesDatabase
Dim mail As EMail

Set currentDB = session.CurrentDatabase

Set mail = New EMail()
Call mail.SetDirectSend(True).SetSender(currentDB.Title & "-DB")
Call mail.SetSendTo("tbahn@assono.de").SetSubject("Dritter Test")
Call mail.SetStyle("headline").AppendText("Dritter Test")
Call mail.SetStyle("standard").AddNewLine(2)
Call mail.AppendText("Direkte Verwendung erlaubt komplexe E-Mails").
Call mail.AddNewLine(2)
Call mail.AppendText("Verknüpfung zur " & currentDB.Title & "-Datenbank: ").
Call mail.AppendDocLink(currentDB, currentDB.Title, "").AddNewLine(1)
Call mail.AddAdditionalMailItem("AktuellerBenutzer", session.UserName)
Call mail.AddAdditionalMailItem("ReplyTo", "tbahn@assono.de")
Call mail.Send()
```

assono

Class EMail – Goldene Mitte

- das Beste aus beiden Welten:
  - einige wenige Prozeduren für die häufigen Standardfälle
  - diese Prozeduren nutzen intern die EMail-Klasse
  - bei Sonderfällen direkte Nutzung der EMail-Klasse

assono

Class EMail – Erweiterungen

- Unterklasse EMailMIME
  - erstellt MIME-E-Mails („HTML-E-Mails“)
- Unterklasse EMailFromTemplate
  - erstellt E-Mails aus E-Mail-Vorlagen
  - Platzhalter werden durch Feldwerte und Formelergebnisse ersetzt
- Klasse SerienEMail
  - nutzt EMailFromTemplate für Serien-E-Mails

assono

Apropos Erweiterungen…

- Man kann Klassen nachträglich erweitern – klar.
  - Nutzer der Klasse müssen höchstens neu kompilieren.
- Man kann Klassen nachträglich ändern, aber:
  - ändert man die „öffentliche“ Schnittstellen müssen Nutzer der Klasse ggf. ihren Code anpassen (böse!);
  - ändert man nur die Implementierung der öffentlich bekannten Methoden oder nur interne Attribute und Methoden müssen Nutzer nichts anpassen.
- Generell **so wenig wie möglich öffentlich** machen, also als Public deklarieren!

assono

Testen – Unit Tests

- Testen ist wichtig, aber manchmal nicht ganz einfach.
- Klassen sind (relativ) abgeschlossene Einheiten, die sich gut isoliert testen lassen.
- Unit Tests: Testen von „Einheiten“, wie z. B. Klassen

assono

Testen – Unit Tests (forts.)

- Schreibe Test-Code, z. B. Agenten, die
  - die ggf. notwendigen Voraussetzungen schaffen
  - Objekte erstellen, Methoden aufrufen
  - Ist-Zustand danach mit Soll-Zustand vergleichen
  - ggf. erzeugte Test-Dokumente wieder löschen
  - Abweichungen zwischen Soll und Ist ausgeben
- Nach wesentlicher Änderung den Test-Agent starten und gleich sehen, ob etwas nicht (mehr) funktioniert.

assono

Fragen?

- jetzt stellen – oder später:

tbahn@assono.de
www.assono.de/blog
04307/900-401

assono
IT-Consulting & Solutions

- Folien unter:
  www.assono.de/blog/d6plinks/EC-2015-OOP