Objektorientierung in LotusScript effizient einsetzen

Bernd Hort
bernd.hort@hort-net.de

Agenda

- Vorstellung
- Motivation
- Objektorientierung in LotusScript
- Design-Prinzipien
- Vorstellung Beispiel-Anwendung
- Performance
- Tools
- Tipps & Tricks
- Fragen & Antworten

Vorstellung

- Bernd Hort
- Diplom-Informatiker
- Lotus Notes Anwendungsentwicklung seit 1995
- IBM Certified Application Developer - Lotus Notes and Domino 7
- IBM Certified System Administrator – Lotus Notes and Domino 7
- IBM Certified Instructor SA & AD – Lotus Notes and Domino 7

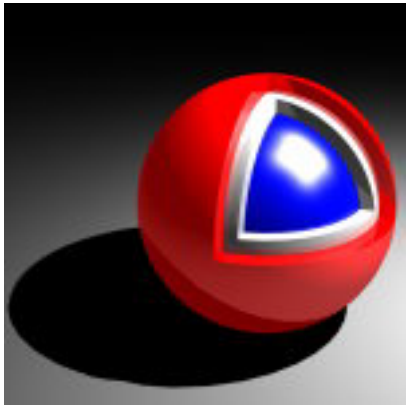
Was ist OO-Programmierung?

- Eine Vorgehensweise, um große und komplexe Systeme intuitiv zu entwickeln.
- Ein Ansatz, die reale Welt abzubilden.
- Fokusiert auf

  Daten & Aktivitäten

  (Properties) (Methods)

- Implementierungsdetails verbergen
  => Information Hiding



Wartbarkeit

Für wen schreiben Sie Ihren Code?

Für die Person, die Ihren Code warten muss!

SIE!

Wartbarkeit

- Zusammengehöriger Code befindet sich an einer Stelle
- Die einzelnen Methoden und Properties sind kürzer und übersichtlicher
- Durch die Kapselung kann die interne Implementierung verändert werden, ohne das es Auswirkungen auf aufrufende Funktionen hat.

Objektorientierung in LotusScript

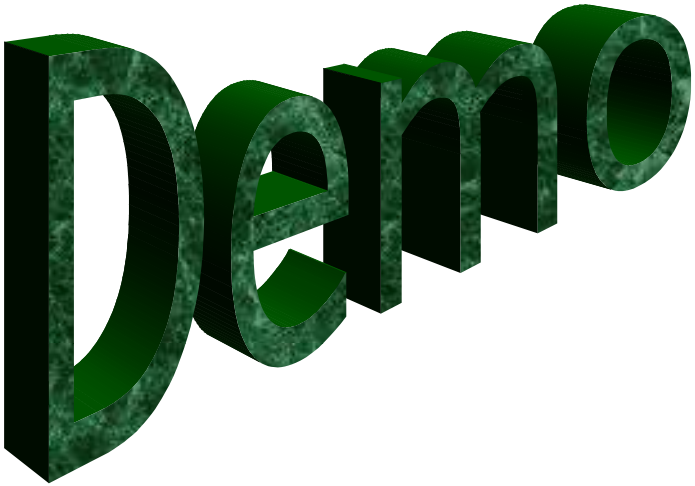
Definition einer Klasse

```
Class Company                                                          Klasse
        Private strCompanyName As String                               Membervariablen
        Private strCompanyNr As String
        Public ContactName As String

        Sub New (strCompanyName As String, strCompanyNr As String)
                Me.strCompanyName = strCompanyName
                Me.strCompanyNr = strCompanyNr
        End Sub 'Contact.New                                           Konstruktor

        Property Get CompanyName As String                             Property Get
                CompanyName = Me.strCompanyName
        End Property

        Property Set CompanyName As String                             Property Set
                Me.strCompanyName = CompanyName
        End Property

End Class
```

Verwenden von Klassen

```
Dim objCompany As Company 'Variable definieren

Set objCompany = New Company ("DaimlerChrysler", "DC")
'Objekt intialisieren
objCompany.ContactName = "Dr. Dieter Zetsche" 'Setzen der
öffentlichen Membervariablen

objCompany.CompanyName = "DaimlerChrysler AG" 'Setzen
der privaten Membervariaben

Messagebox objCompany.ContactName & " - " &
objCompany.CompanyName & " (" &
objCompany.CompanyNr & ")", 64, "Demo" 'Zugriff auf die
Membervariablen
```

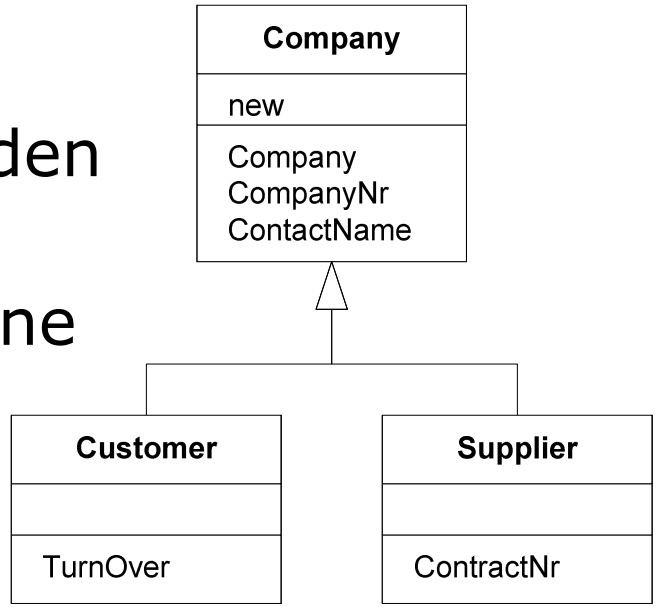
Demo einfache Klasse



«Agent»
Demo\Company Class

Vererbung

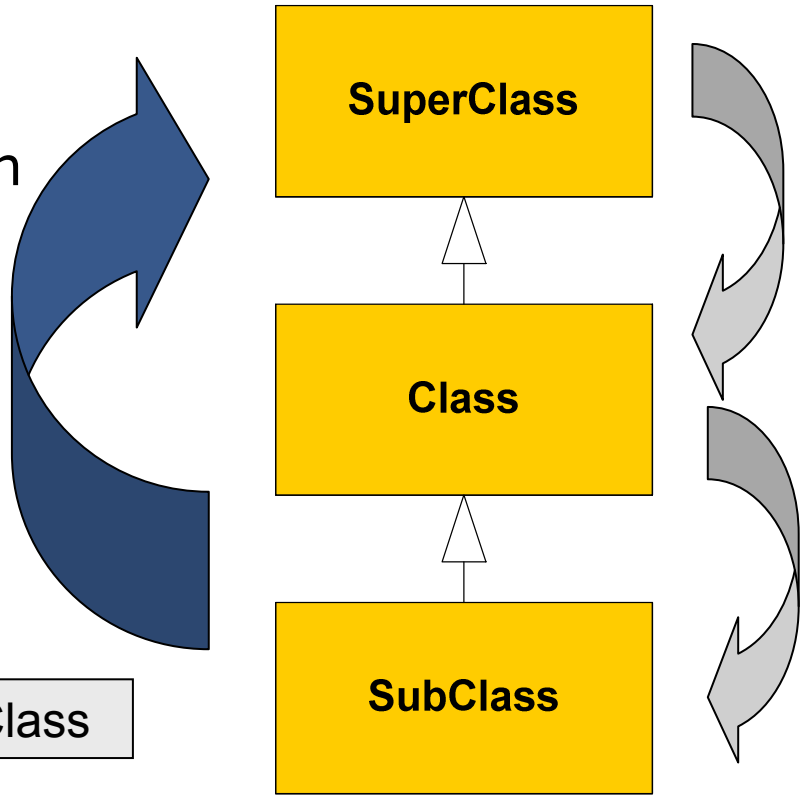
- Übernehmen von Properties und Methoden einer Klasse
- Ergänzung durch eigene Properties und Methoden




```
Class Customer As Company
        Public TurnOver As Double

        Sub New (strCompanyName As String, strCompanyNr As String)

        End Sub 'Customer.New

End Class 'Customer
```

Initialisierungsreihenfolge bei Unterklassen

- Es wird zuerst der Konstruktor der Superklasse aufgerufen
- Danach alle weiteren Konstruktoren




```
Set objSubClass = New SubClass
```

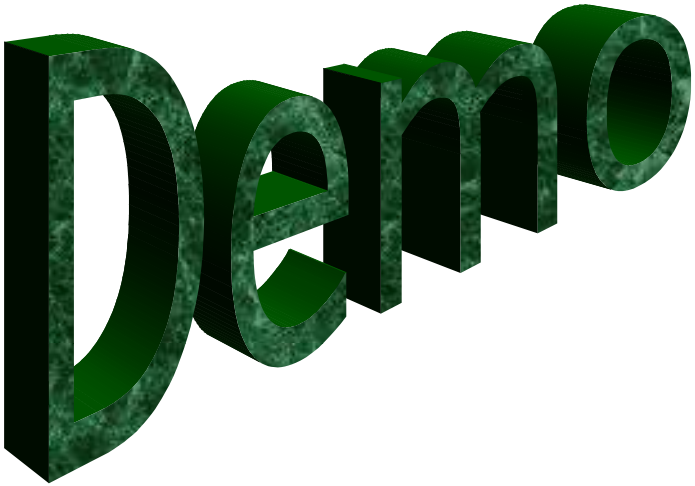
Konstruktur überschreiben

- Abgeleitete Klassen können einen von der Elternklasse abweichenden Konstruktur haben.

```
Class Supplier As Company
        Private strContractNr As String

        Sub New (strCompanyName As String, strCompanyNr As String, _
        strContractNr As String), _
        Company (strCompanyName, strCompanyNr)

                Me.strContractNr = strContractNr

        End Sub 'Supplier.New

End Class 'Supplier
```

Übergabe der Parameter für die Elternklasse

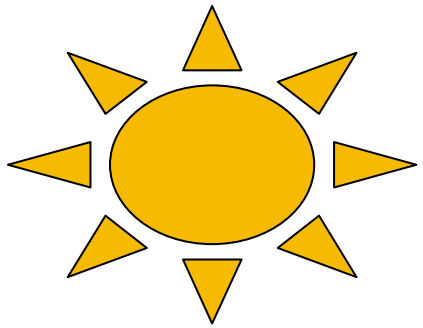
Demo abgeleitete Klasse



«Agent»
Demo\Customer/Supplier Class

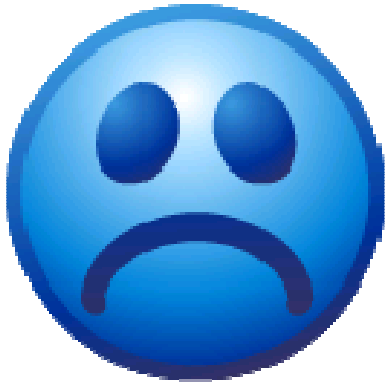
Methoden und Properties überschreiben

- In der abgeleiteten Klasse können Methoden und Properties mit gleichem Namen definiert werden.
- Die Signature muss gleich bleiben.
- Methoden und Properties der Elternklasse können mit `Elternklasse..MethodenName` aufgerufen werden



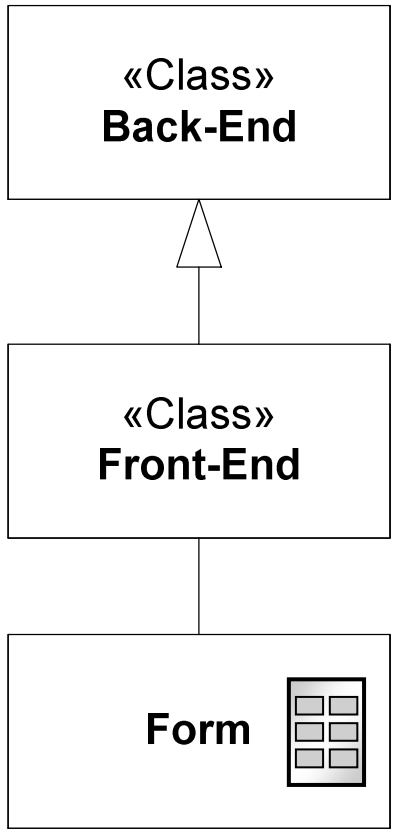
Standard Notes-Klassen können nicht abgeleitet werden

- Es ist nicht möglich, die Standard Notes-Klassen abzuleiten!



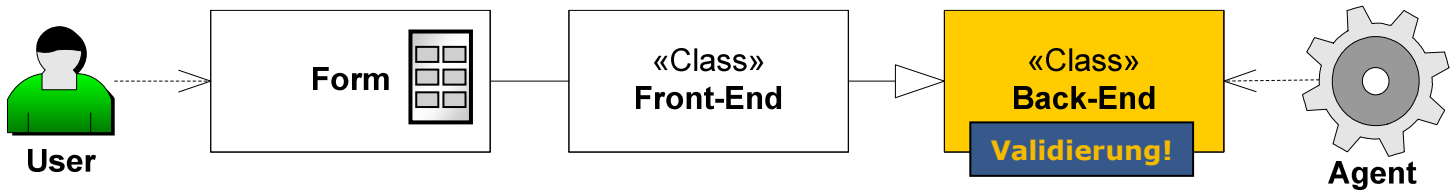
Design Prinzipien

- Trennung von Back-End- und Front-End-Klassen
- Vererbung stellt die Back-End-Methoden auch im Front-End zur Verfügung
- Verlagerung möglichst viel Funktionalitäten in das Back-End
  - Insbesondere Eingabe-Validierung und Plausibilätsprüfung



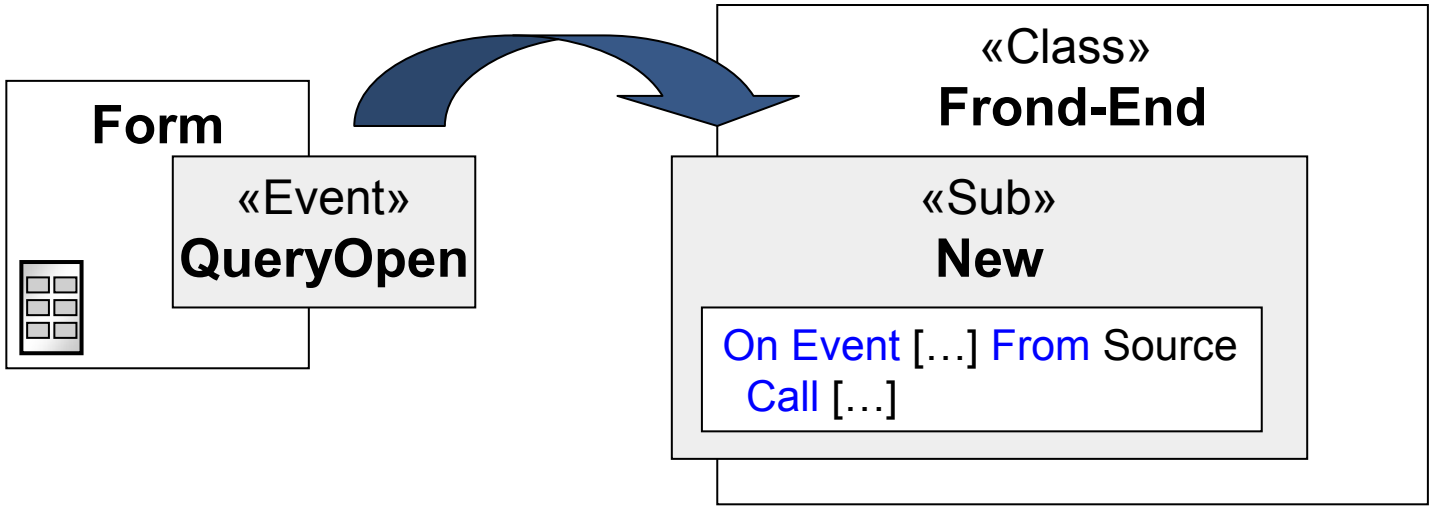
Vorteile der Verlagerung der Eingabe-Validierung ins Back-End

- Ein zentraler Punkt an dem alle Regeln für die Gültigkeit eines Dokumentes hinterlegt werden
- Verwendung der gleichen Überprüfungs-routinen sowohl für Agents als auch für den Anwender



Form-Event-Handling in Front-End-Klasse

Ziel ist es, möglichst wenig Code in der Maske selber zu haben.



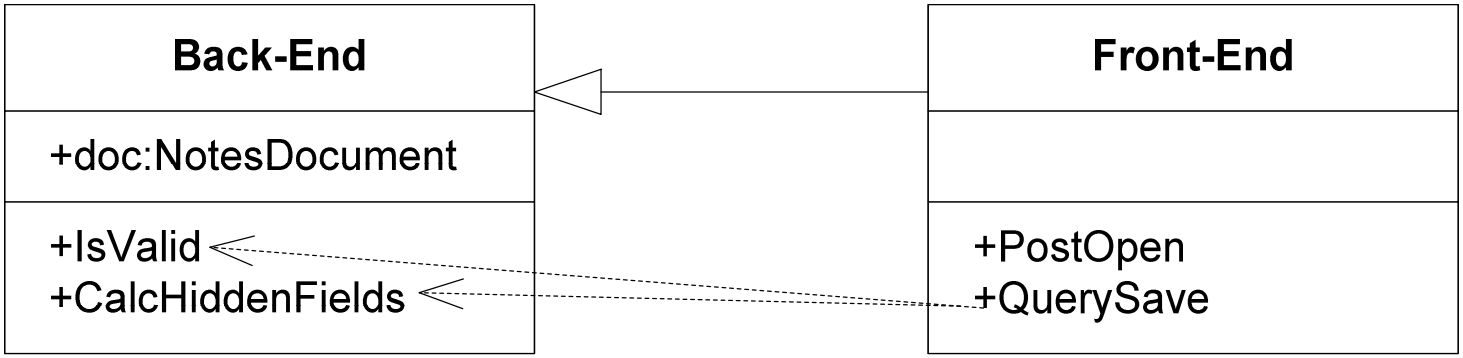

```
Sub Queryopen(Source As Notesuidocument, Mode As Integer, Isnewdoc As Variant,
Continue As Variant)
        Set objProjectUI = New ProjectUI(Source, Mode, Isnewdoc, Continue)
End Sub
```

Zuordnung Event-Handling

```
Sub New (Source As NotesuiDocument, Mode As Integer, IsNewDoc As Variant,
Continue As Variant), _
Project (Source.Document, objErrorContainer)

        On Event PostOpen From Source Call PostOpen
'       On Event QueryModeChange From Source Call QueryModeChange
'       On Event PostModeChange From Source Call PostModeChange
'       On Event PostRecalc From Source Call PostRecalc
        On Event QuerySave From Source Call QuerySave
'       On Event PostSave From Source Call PostSave
'       On Event QueryClose From Source Call QueryClose

        If Not objErrorContainer Is Nothing Then
                Call ErrorProcessingUI(objErrorContainer, doc)
                Exit Sub
        End If

End Sub 'ProjectUI.New
```

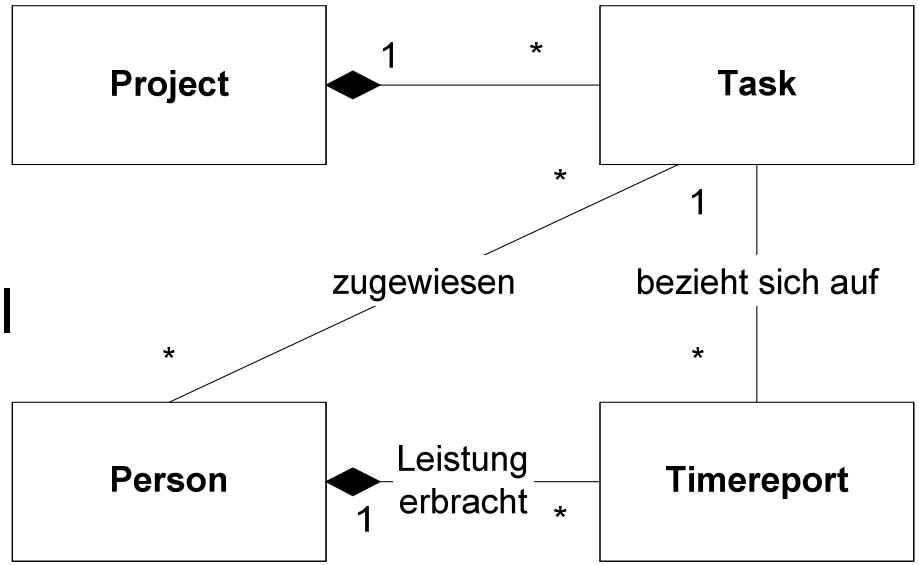
Grundgerüst

- Alle meine dokument-zentrierten Klassen haben ein gemeinsames Grundgerüst.

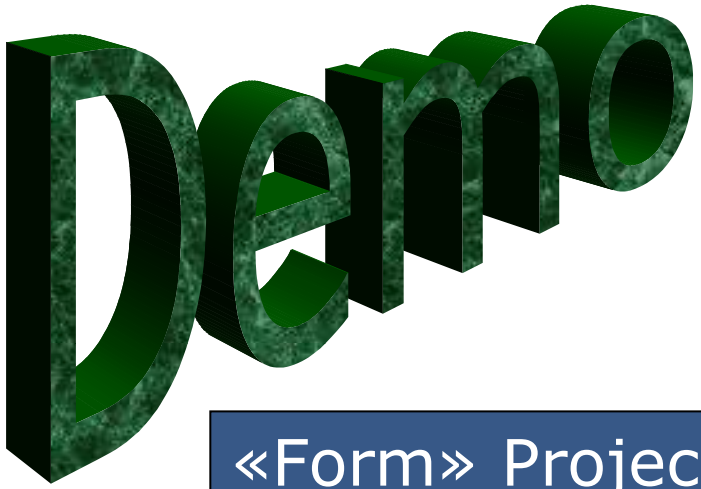


Beispiel-Anwendung

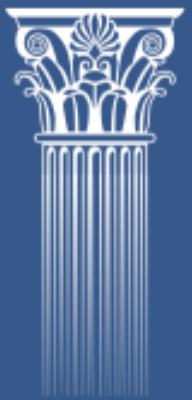
- Simple Projektverwaltung
- Bestandteile
  - Projekt
  - Aktivität
  - Person
  - Stundenzettel



Demo Beispiel-Anwendung



«Form» Project
«Form» Task



Fehler-Handling

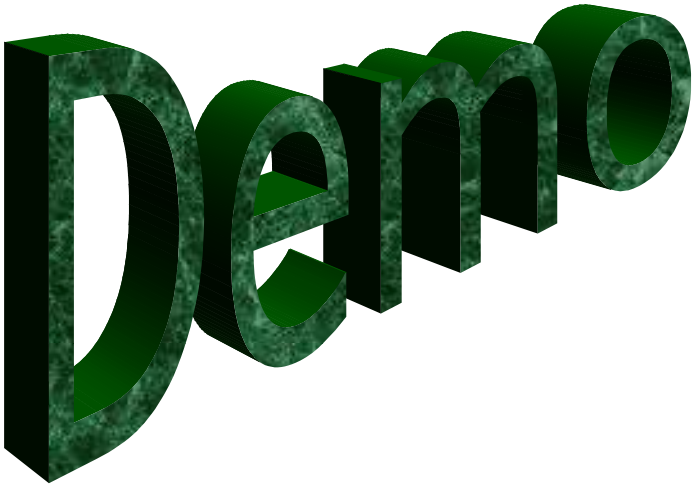
- Eine Klasse um Fehler-Details aufzunehmen
- Im Back-End wird der Fehler an die aufrufende Procedure übergegeben
- Im Front-End wird der Anwender per Dialog informiert
- Agenten senden den Fehler direkt an die Administratoren

«Back-End» **ErrorContainer**
+Context:String +ErrNr:Integer +ErrorDescription:String +ErrorMessage:String +ErrorPrintLine:String +ErrorTrace:String +ErrorTraceARRAY:Variant +LineNumber:Integer +Procedure:String
+New(Err:Integer, Erl:Integer, Error:String) +Send(docLink:NotesDocument)

Function LSI_Info

- Undokumentiert, aber seit R5.x vorhanden
- Parameters (Soweit bekannt)
  - LSI_Info(1) = Zeile
  - LSI_Info(2) = Function oder Sub
  - LSI_Info(3) = Module
  - LSI_Info(6) = LotusScript - Version
  - LSI_Info(9) = Sprache (e.g. en for English)
  - LSI_Info(12) = Name der aufrufenden Procedure (the “calling code ”)
  - LSI_Info(14) = Hierarchie der Proceduren-Aufrufe (ab 6.x)
  - LSI_Info(50) = Memory Allocated
  - LSI_Info(51) = Memory Allocated vom OS
  - LSI_Info(52) = Memory Blocks Used

Demo Fehlerhandling



«Agent»
Errorhandling\Demo UI

Performance

- Klassen bedeuten einen gewissen Overhead
- Bei großen Schleifen löschen Sie nicht mehr benötigte Objekte mit `Delete Objekt`
- Das Laden vieler kleiner Script Bibliotheken verlangsamt das Öffnen von Masken

➢ Dynamisches Laden von Script Bibliotheken

Dynamisches Laden von Script Bibliotheken - Referenzen

- Bill Buchan – Lotusphere 2005
  - BP107 Best Practices for Object Oriented LotusScript
- Beschrieben in dem Redbook
  - “Performance Considerations for Domino Applications”
  - SG24-5602
  - Appendix B, Page 243
- Gary Devendorf Web Services Beispiele

„Factory“-Klasse zum Erzeugen von Objekten

- Neben der eigentlichen Klasse wird eine Factory-Klasse benötigt
- Die Objekt-Variable muss als Variant definiert werden

```
Class Customer
        Sub New (objErrorContainer As ErrorContainer)

        End Sub 'Customer.New
End Class 'Customer
```

```
Class CustomerFactory
        Public Function Produce(objErrorContainer As ErrorContainer) As Variant
                Set Produce = New Customer(objErrorContainer)
        End Function
End Class 'CustomerFactory
```

```
Dim objCustomer as Variant
Set objCustomer = CreateClass(".AppCustomerClass", "Customer", objErrorContainer)
```

„Dynamisches Laden“ – Die Magie

Verwendung von Execute zusammen mit einer globaler Variablen

```
Public newObject As Variant 'Global definiert
```

```
Function CreateClass (strScriptLibraryName As String, strClassName As String,
objErrorContainer As ErrorContainer) As Variant
        Dim strExecute As String

        strExecute = _
        |
        Use "| & strScriptLibraryName & |"
        Sub Initialize
          Set newObject = New | & strClassName & |Factory
        End Sub
        |

        Execute strExecute 'Der Code im String wird ausgeführt

        Set CreateClass = newObject.Produce(objErrorContainer)

End Function
```

Konsequenzen „Dynamisches Laden“

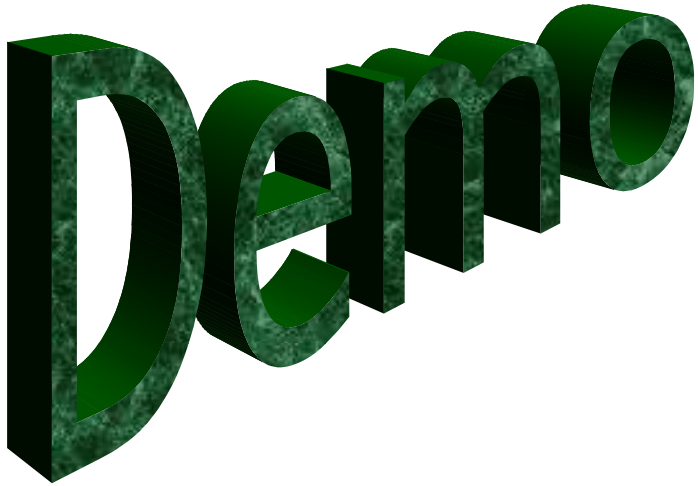
Positive Konsequenzen

- Script Bibliotheken werden nur noch geladen, wenn sie benötigt werden.
- Deutlich schnelleres Öffnen von Masken
- Während der Laufzeit können in Abhängigkeit der Plattform oder Version unterschiedliche Klassen geladen werden!

Negative Konsequenzen

- Keine Überprüfung mehr, ob Klasse, Methoden und Properties vorhanden sind

Demo „Dynamisches Laden“



«ScriptLibrary» .AppTaskClass:
«Method» Task.UpdateResponses

«ScriptLibrary» .AppProjectClass:
«Method» Project.UpdateResponses

Tools – LotusScript.doc

- Generiert eine web-basierte Dokumentation
- Ähnlich wie JavaDoc
- Unterstützt zusätzliche Kommentare
- http://www.lsdoc.org
- Kostenlos!

Tools - GhostTyper

- Einfügen von Code Snippets direkt aus dem Domino Designer heraus
- http://www.ghosttyper.de
- Kostet ca. 35,- €
- Meine GhostTyper-Archive können unter http://www.hort-net.de/tools.html heruntergeladen werden
- 5% Rabatt bei Bestellung über meine Website

Tools – Teamstudio Script Browser

- Zeigt alle Subs, Functions und Classes in einer Datenbank
- Analysiert die Referenzen
- http://www.teamstudio.com/support/scriptbrowser.html
- Kostenlos!
- Weitere freie Tools im Blog von Craig Schumann / Chef-Entwickler von Teamstudio
  - http://blogs.teamstudio.com

Tools - LS Class Buddy

- Navigation innerhalb der Klassen einer Script Bibliothek
- http://www.ddextensions.com/lsclassbuddy.html
- 39,50 $
- Sorry, kein Rabatt

Tools – Formatierung von LotusScript in HTML / RTF

- Das für diese Präsentation verwendete Code Coloring
- nsf tools
- http://www.nsftools.com/tips/NotesTips.htm#ls2html
- Kostenlos!
- Sehr guter Blog von Julian Robichaux
  - http://www.nsftools.com/blog

Tipps & Tricks

- Fangen Sie mit meinen Code Snippets an
- Überlegen Sie beim Entwickeln, welche Klassen das Potential für Re-Use haben
- Verwenden Sie eine Namens-Konvention
- Starten Sie mit einem kleinen Projekt
- Geben Sie sich ein wenig Zeit

Übertreiben Sie es nicht!

- Ja, Sie können jede Klasse von einer einzelnen Basisklasse ableiten lassen.
- Ja, Sie können in diese Basisklasse jede Menge Methoden integrieren, von denen Sie schon immer glaubten, sie haben zu müssen.
- Nein, Ihr Code wird dadurch nicht schneller. ;-)
- Nein, Ihr Code wird dadurch nicht wartungsfreundlicher. ;-)

Noch Fragen?

Kontakt / Feedback

Bernd Hort/Hort-Net IT Consulting

Dockenhudener Chaussee 21B
D-25469 Halstenbek

Tel. 04101 / 48747

bernd.hort@hort-net.de
http://www.hort-net.de

Download der Folien und Beispiele
http://www.hort-net.de/EntwicklerCamp

Yellow is the new black