assono

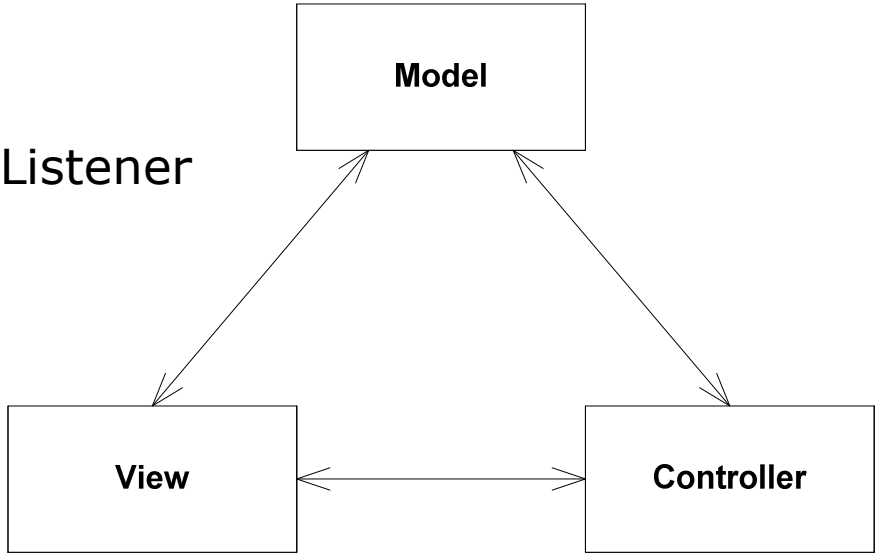
OOP in LotusScript – der nächste Schritt

Aufbau eines MVC-Frameworks

Bernd Hort
assono GmbH
bhort@assono.de
http://www.assono.de
+49 (0)177 / 44 487 47

Agenda

- Vorstellung
- Motivation
- Model View Controller Pattern
- «Class» BaseModel
- «Class» BaseController
- «Class» AbstractChangeListener
- «Class» ItemLevelHistoryChangeListener
- Relationen
- 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

Certified for IBM Lotus e-business software

Warum das Ganze?

Früher Feierabend!

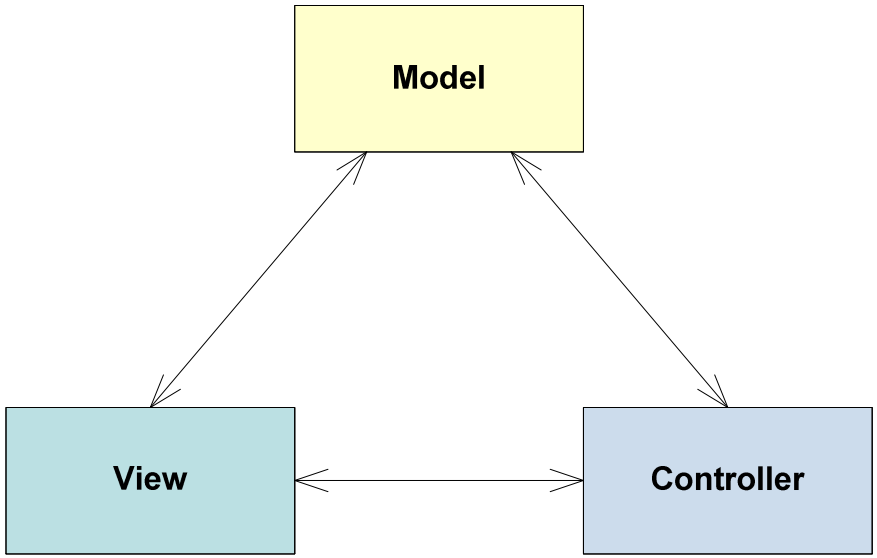
…und im Ernst?

- Entwicklung auf einer höheren Abstraktionsebene
- Bewältigung von Komplexität
- Höhere Wiederverwendbarkeit
- Weniger Fehleranfällig
- Bessere Wartbarkeit

- Es jetzt lernen, weil es in allen OO-Sprachen verwendet wird.

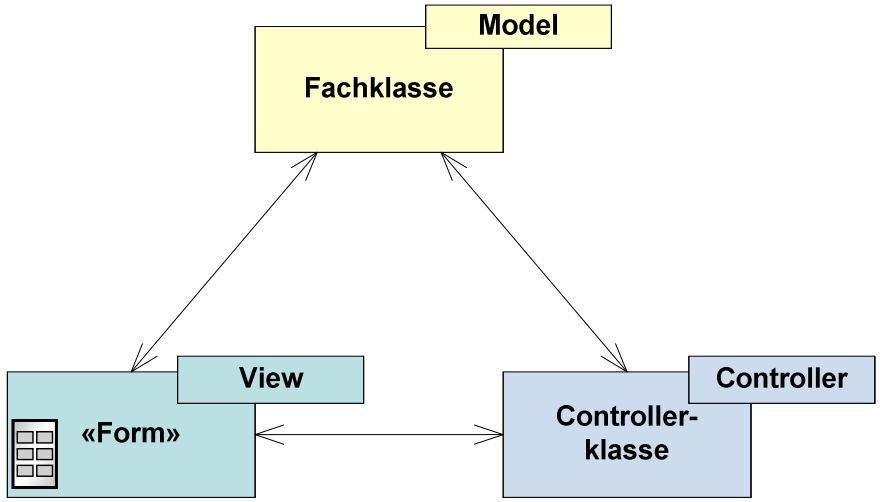
Model View Controller Pattern

- Ursprünglich aus Smalltalk
- Allgemeines OO-Prinzip für die Entwicklung von GUIs
- Trennung von Fachklassen und deren Darstellung
  - *Model* – Fachklasse
  - *View* – Darstellung
  - *Controller* – Benutzerinteraktion
- Die Fachklasse weiß nichts von seiner Darstellung!



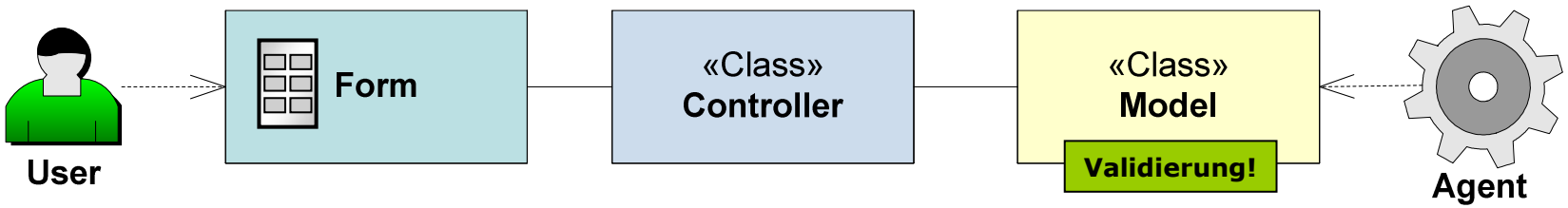
Model View Controller in Notes

- Die Maske stellt die Daten dar und nimmt die Benutzereingabe entgegen.
- Alle fachlichen Anforderungen werden in der Fachklasse implementiert.
- Die Controllerklasse ist das Verbindungsstück zwischen der Fachklasse und der Maske.



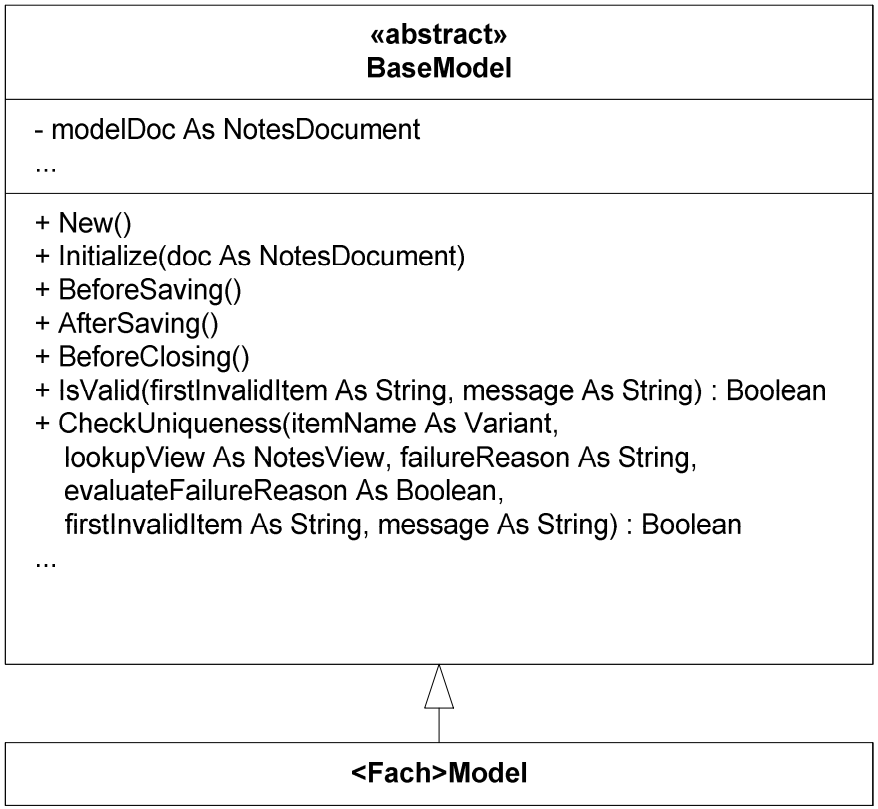
Umsetzung aller fachlichen Anforderungen in der Fachklasse

- Jegliche fachliche Anforderung sollte auch in der Fachklasse umgesetzt werden.
- Das beinhaltet insbesondere auch alle Eingabevalidierungen und Plausibilitätsüberprüfungen.
- Die Fachklasse wird so implementiert, dass sie keine UI-Methoden verwendet.
- Somit können die gleichen Überprüfungsroutinen sowohl für Agents als auch für Benutzereingaben verwendet werden.



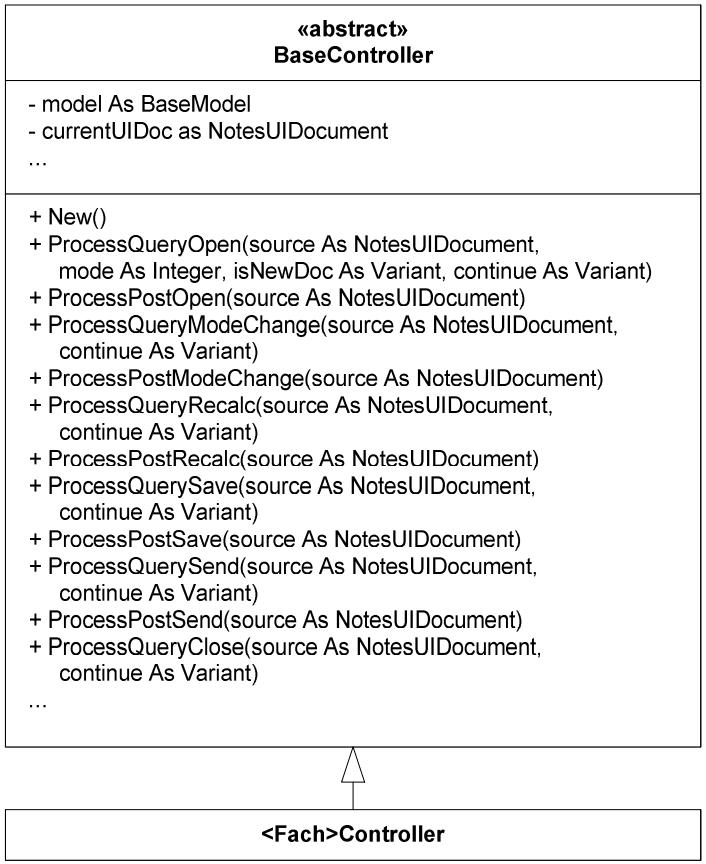
Abstrakte «Class» BaseModel

- Basis für alle fachlich motivierten Klassen
- Die meisten Methoden sind nur Schnittstellen-Definitionen, d.h. der Methodenrumpf ist leer
- Keine UI-Methoden bzw. –Klassen erlaubt
- Die eigentliche Fachklasse wird davon abgeleitet



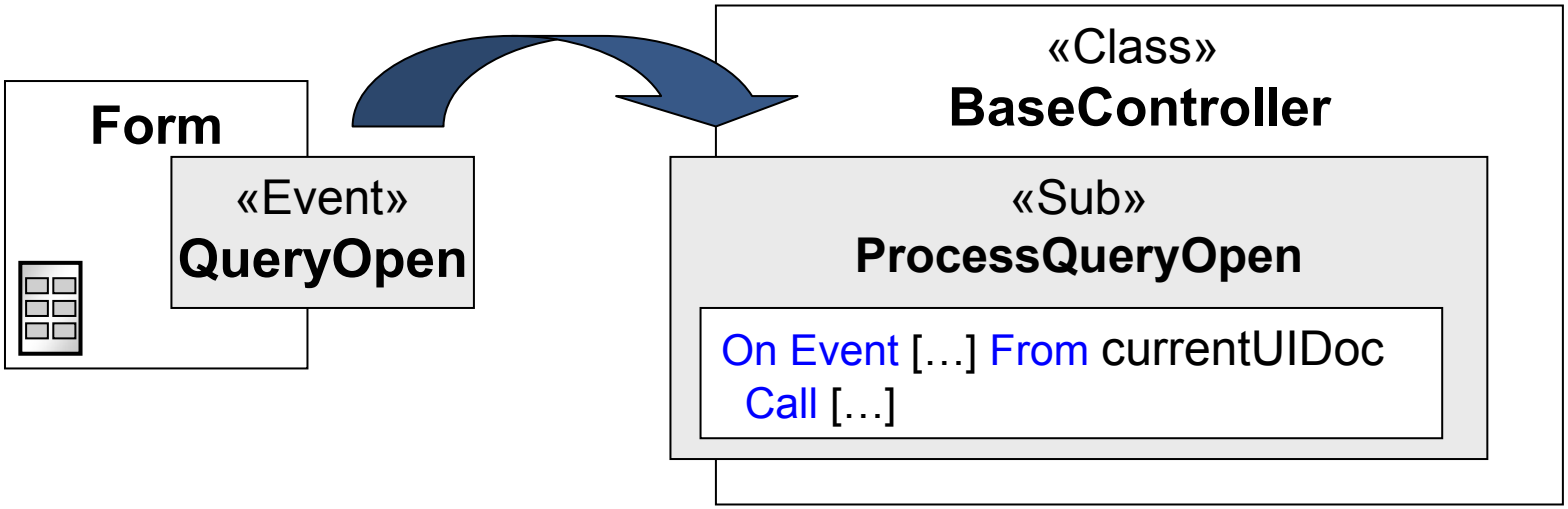
Abstrakte «Class» BaseController

- Schnittstelle zwischen Fachklasse und Maske
- Abfangen aller Events in der Maske



Form-Event-Handling in Controller Class

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



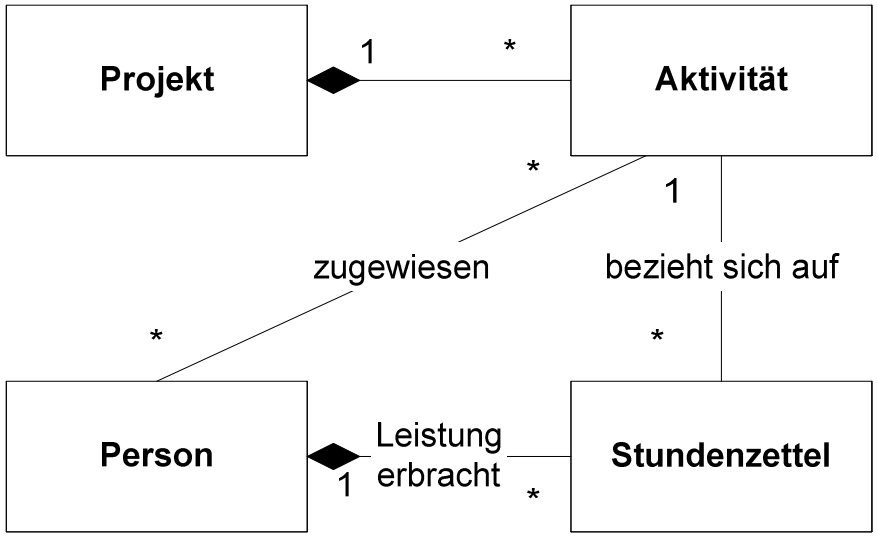

```
Sub Queryopen(Source As Notesuidocument, Mode As Integer, Isnewdoc As Variant,
Continue As Variant)
        Call CreateProjectController(source, mode, isNewDoc, continue)
End Sub
```

Zuordnung Event-Handling

```
Public Sub ProcessQueryOpen(source As NotesUIDocument, mode As Integer, isNewDoc As Variant, continue As Variant)

	If Not IsDebugMode Then On Error Goto errorHandler

	Set Me.currentUIDoc = source
	Me.isNewDocument = Cbool(isNewDoc)

	continue = continue And (currentDB.CurrentAccessLevel >= ACLLEVEL_AUTHOR Or mode = 0) ' do not open in edit mode, if user has not at
	least author access

	If continue Then
		If Not Me.isNewDocument Then ' source.Document is valid for existing documents
			Call model.initialize(source.Document) ' thus we can initialise the model now
			Set currentDoc = model.GetModelDoc()
		End If

		' register all event handlers
		On Event PostOpen From currentUIDoc Call processPostOpen
		On Event QueryModeChange From currentUIDoc Call processQueryModeChange
		On Event PostModeChange From currentUIDoc Call processPostModeChange
		On Event QueryRecalc From currentUIDoc Call processQueryRecalc
		On Event PostRecalc From currentUIDoc Call processPostRecalc
		On Event QuerySave From currentUIDoc Call processQuerySave
		On Event PostSave From currentUIDoc Call processPostSave
		On Event QuerySend From currentUIDoc Call processQuerySend
		On Event PostSend From currentUIDoc Call processPostSend
		On Event QueryClose From currentUIDoc Call processQueryClose
	End If
	Exit Sub

errorHandler:
	If HandleError() Then Resume Next
	End
End Sub ' BaseController.ProcessQueryOpen
```

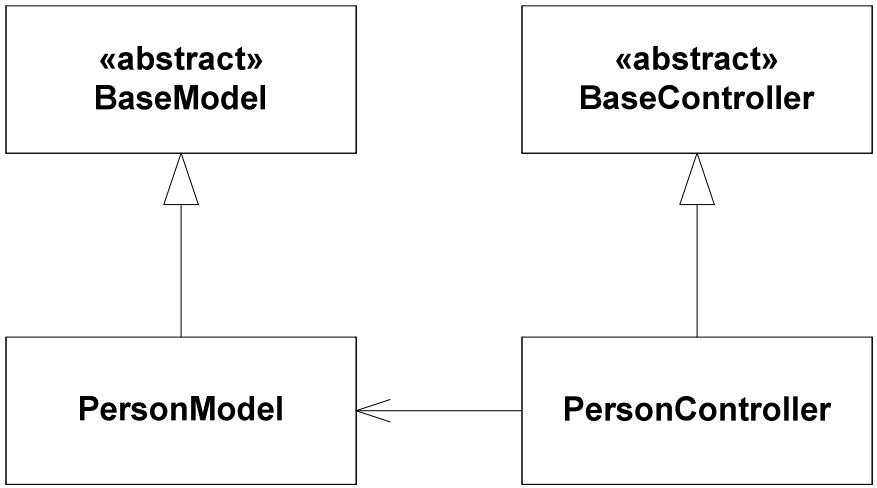
Beispiel-Anwendung

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

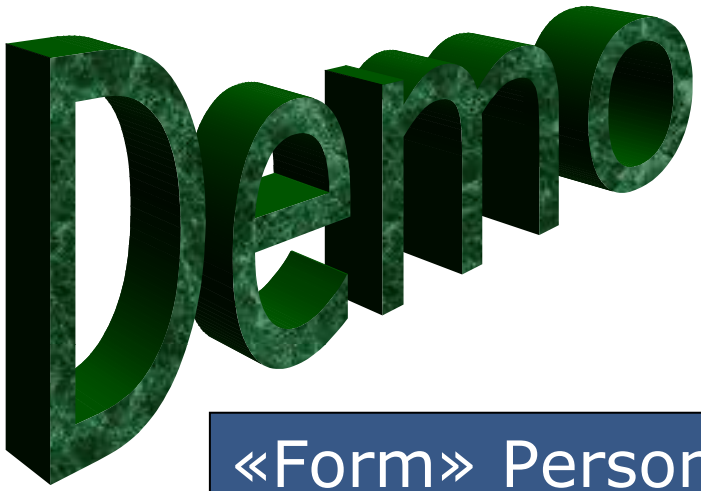


Person

- Fachliche Aspekte in «Class» PersonModel
- Verbindung zur Maske über «Class» PersonController



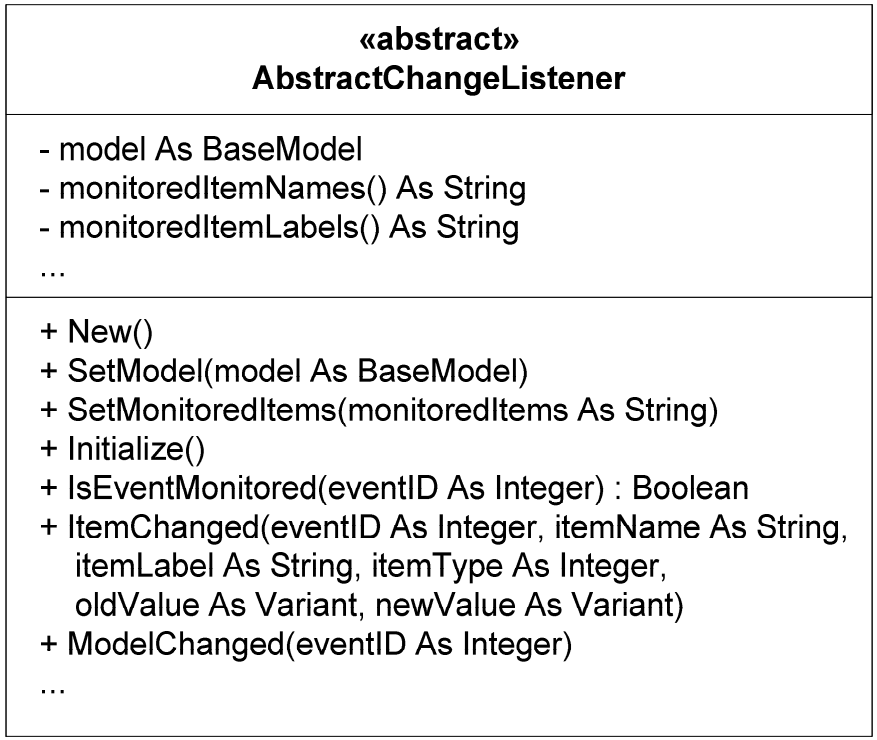
Demo Beispiel-Anwendung



«Form» Person

Überwachung von Änderungen in Feldwerten

- In vielen Fällen gibt es die Notwendigkeit die Inhalte von Feldwerten zu überwachen.
  - Historie
  - Aktualisierung von abhängigen Dokumenten
  - Benachrichtigung wenn Grenzwerte überschritten werden
  - ...
- Statt den gleichen Code immer wieder zu schreiben, die Funktionalität in eigener Klasse gekapselt.

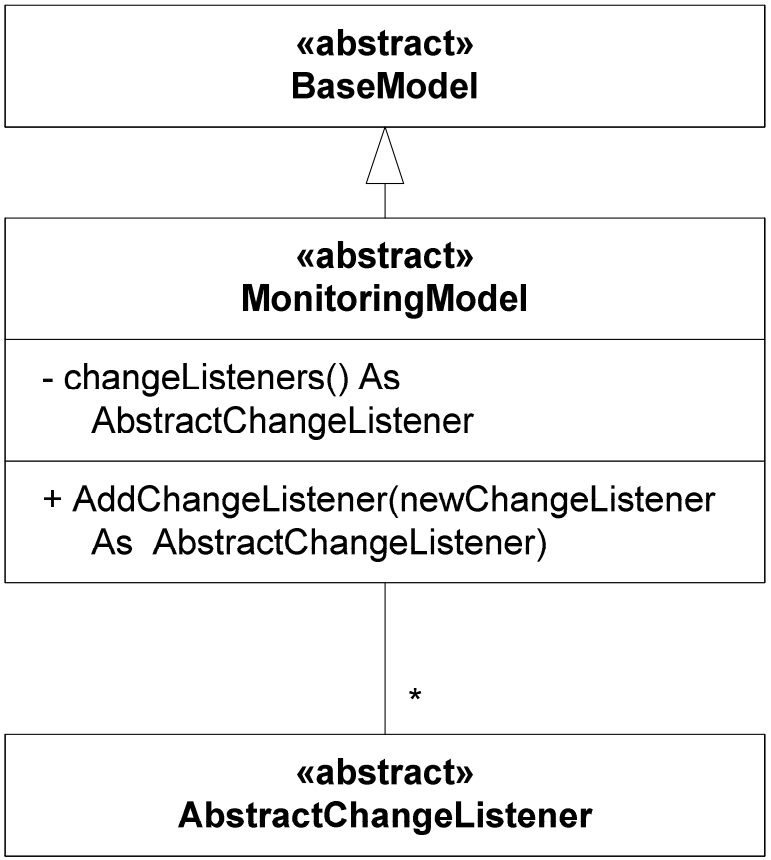


«Class» AbstractChangeListener – zu überwachende Felder

- Mit
  SetMonitoredItems(monitoredItems As String)
  werden die zu überwachenden Felder bestimmt
- Als Parameter wird ein String mit einer durch Kommas getrennten Liste von Feldnamen in dem Format
  Label|Feldname
  oder nur
  Feldname
- z.B. "Personalnummer | MitarbeiterID, Name | MitarbeiterName, Nachname, Vorname"

Abgeleitete «Class» MonitoringModel

- «Class» MonitoringModel abgeleitet von «Class» BaseModel
- Verwaltet die interessierten ChangeListener [0..*]
- Hält intern eine Liste mit zu überwachenden Feldern
- Reagiert auf verschiedene Events
- Benachrichtigt die interessierten ChangeListener bei Änderungen auf
  - Feldebene
  - Dokumentebene



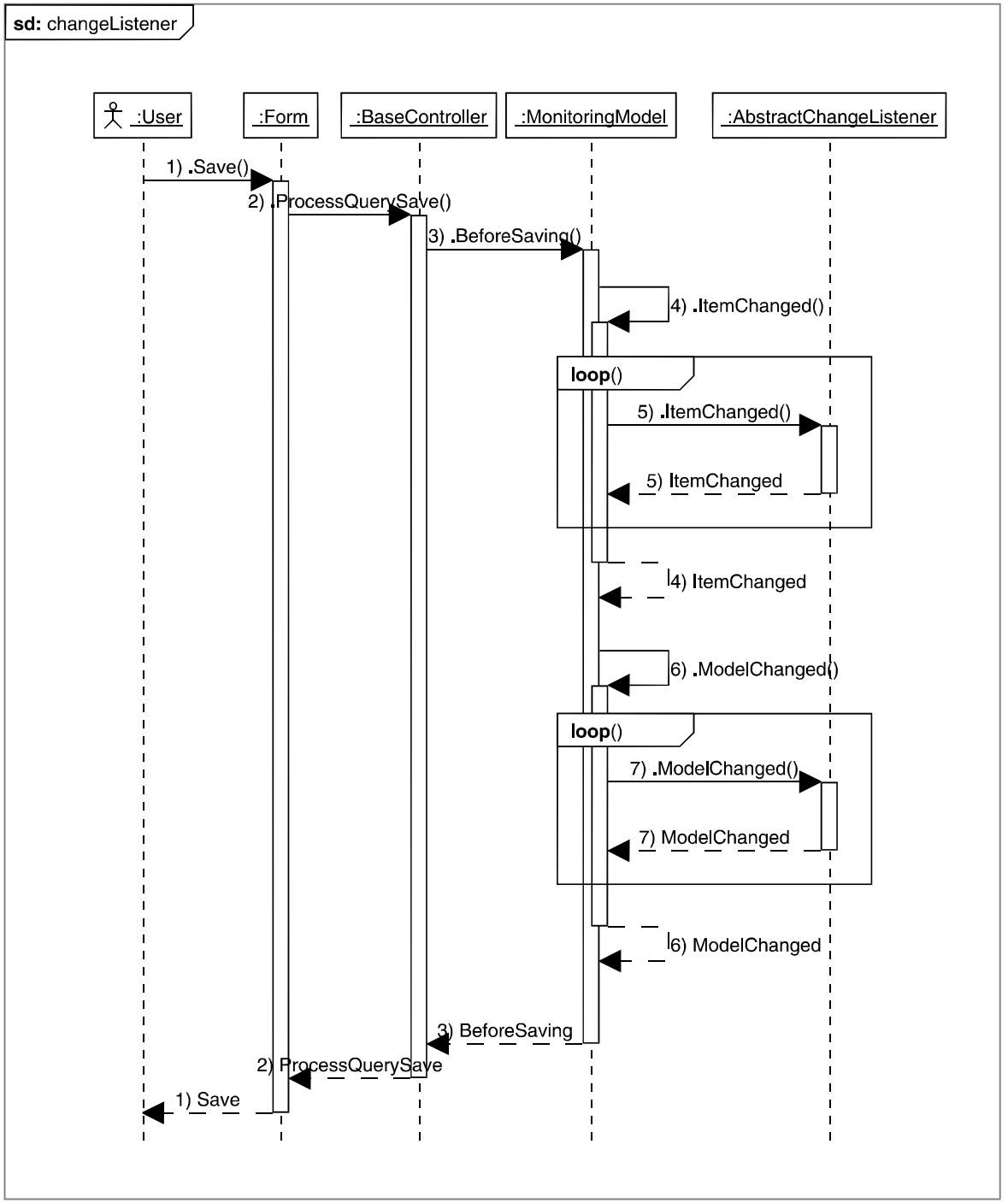
Mögliche Events

- Die folgenden Events werden unterstützt

```
'/**************************************************************************
 ' * ID of "before saving" event (QuerySave).
 '**************************************************************************/
Public Const CHANGE_LISTENER_EVENT_BEFORE_SAVING% = 1

'/**************************************************************************
 ' * ID of "after saving" event (PostSave).
 '**************************************************************************/
Public Const CHANGE_LISTENER_EVENT_AFTER_SAVING% = 2

'/**************************************************************************
 ' * ID of "before closing" event (QueryClose).
 '**************************************************************************/
Public Const CHANGE_LISTENER_EVENT_BEFORE_CLOSING% = 3
```

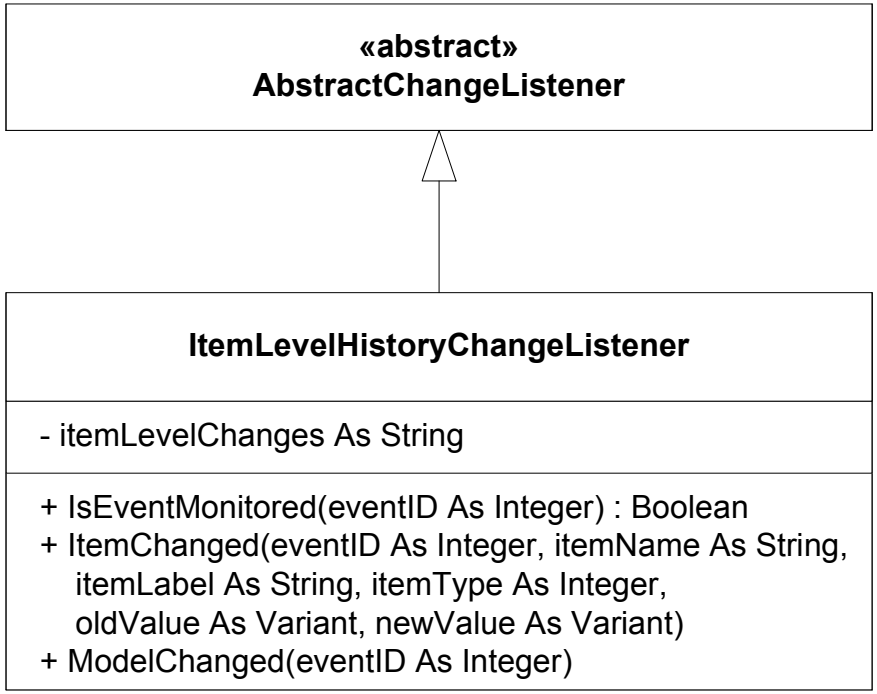
Sequenz-Diagramm ChangeListener

- Durch das Event QuerySave getriggert
- Methode MonitoringModel. BeforeSave()
- Bei Feldänderung Aufruf MonitoringModel. ItemChanged()
- Aufruf .ItemChanged() bei allen an dem Event interessierten ChangeListenern
- Aufruf MonitoringModel. ModelChanged()
- Aufruf .ModelChanged() bei allen interessierten ChangeListenern



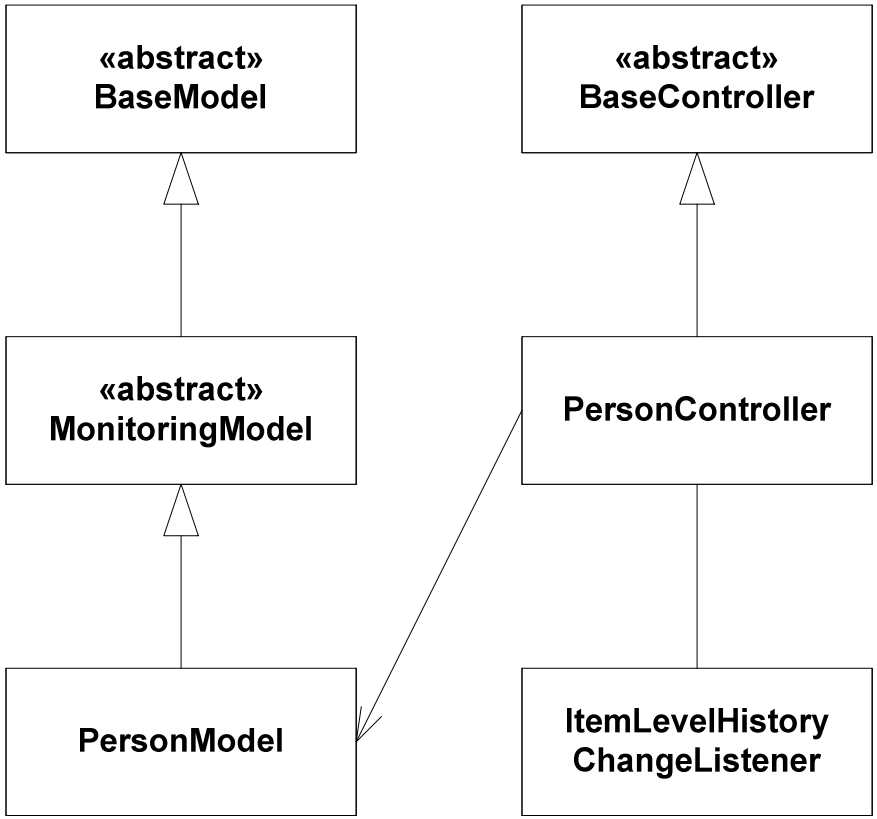
«Class» ItemLevelHistoryChangeListener

- Schreibt Änderungen auf Feldebene in die Historie
- Überlädt die Methoden
  - IsEventMonitored
  - ItemChanged
  - ModelChanged

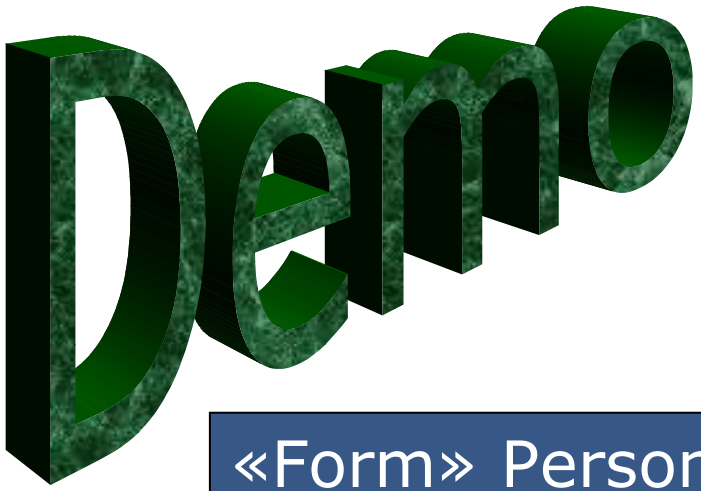


Person – Verwendung ItemLevelHistoryChangeListener

- Elternklasse für «Class» PersonModel in «Class» MonitoringModel ändern
- In der «Class» PersonController die «Class» ItemLevelHistory-ChangeListener eintragen



Demo Beispiel-Anwendung

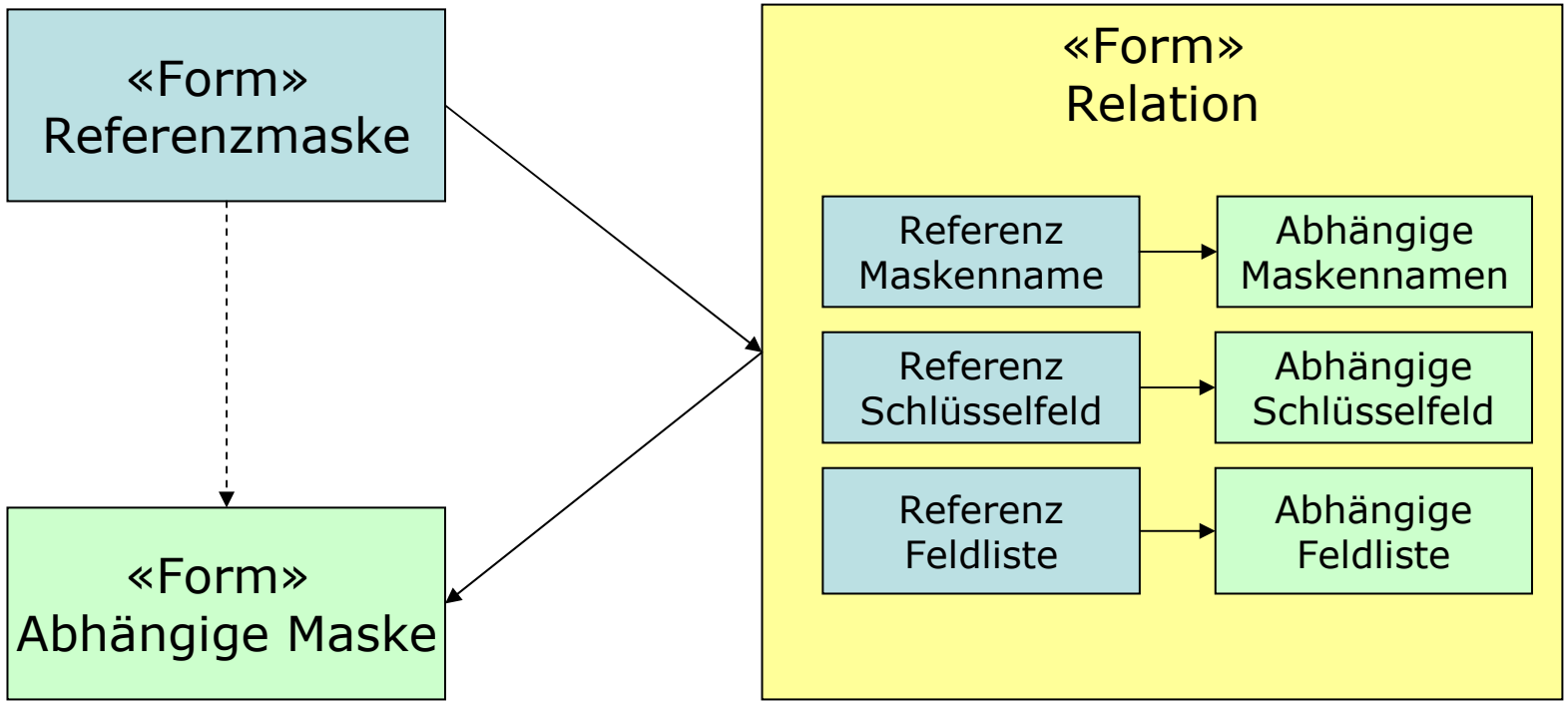


«Form» Person

Relationen

- Die Idee geht auf einen Vortrag von Jens-B Augustiny auf dem EntwicklerCamp 2006 zurück.
- Aktualisierungsabhängigkeiten konfigurierbar
- Vorteile:
  - Leicht erweiterbar
  - Implizite Dokumentation
  - „Dependency Injection“

Relationen


«Form»
Referenzmaske
«Form»
Abhängige Maske
«Form»
Relation
Referenz
Maskenname
Abhängige
Maskennamen
Referenz
Schlüsselfeld
Abhängige
Schlüsselfeld
Referenz
Feldliste
Abhängige
Feldliste

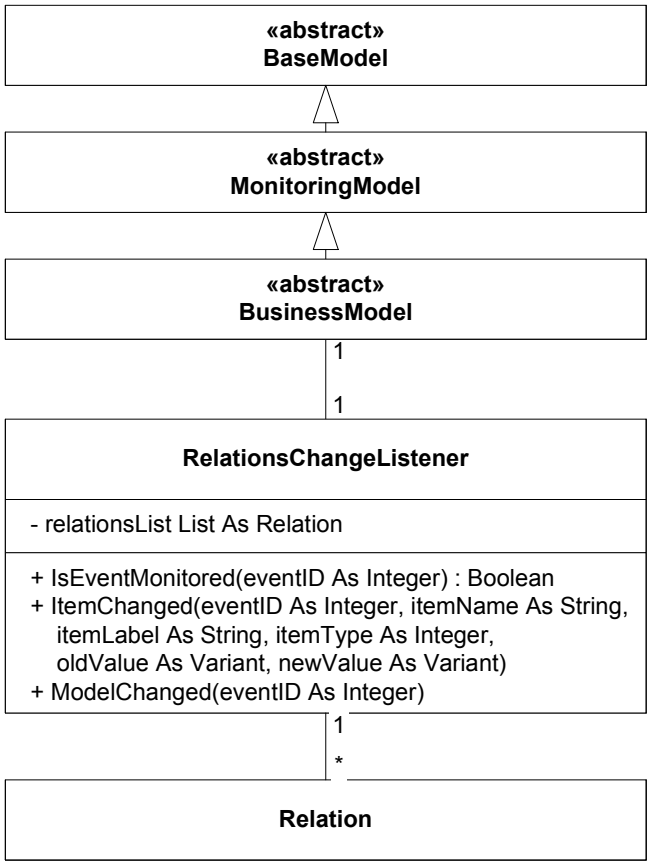
«Class» Relation

- Representiert das Relationen-Dokument
- Aktualisiert die abhängigen Dokumente

Relation
- oldKeyValue As String - monitoredItemNames As Variant - keyItemName As String ...
+ New(relationID As String) + Initialize(relationDoc As NotesDocument) + ReadOldKeyValueFromSourceDoc(sourceDoc As NotesDocument) + IsItemMonitored(itemName As String) + UpdateDependentDocuments(sourceDoc As NotesDocument) ...

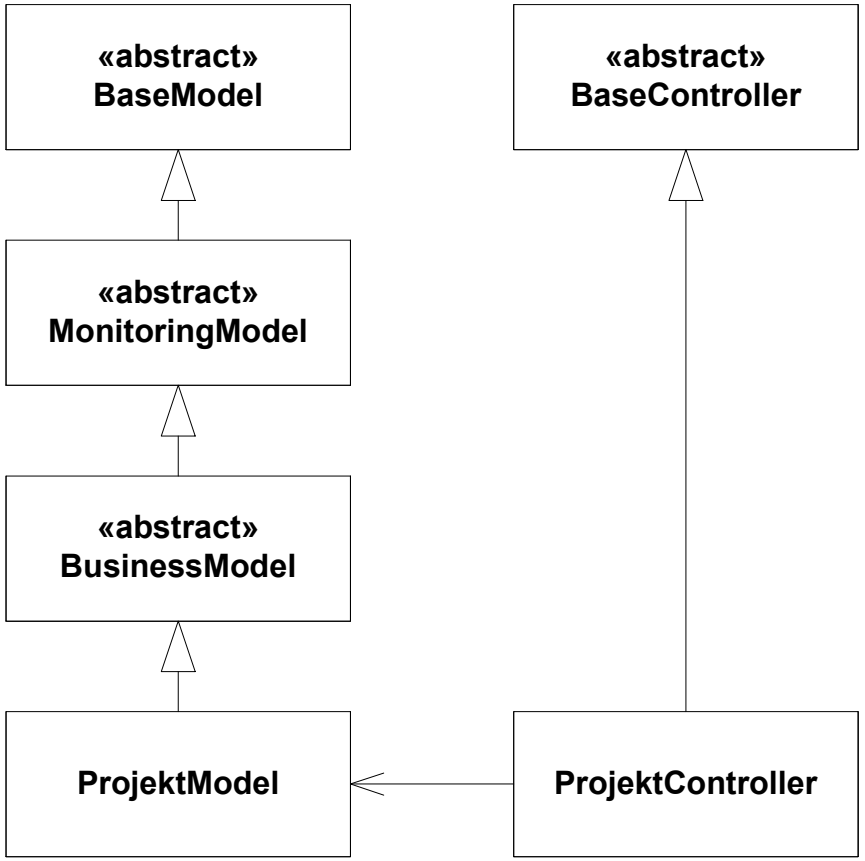
«Class» RelationChangeListener

- Verwaltung aller für eine Maske gefundenen Relationen
- Wird eingebunden über die «Class» BusinessModel

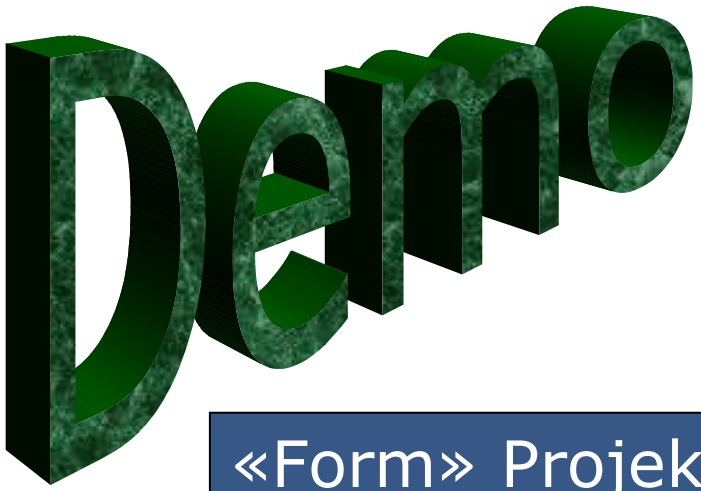


Projekt

- Fachliche Aspekte in «Class» ProjektModel
- Abgeleitet von «Class» BusinessModel um von Relationen Gebrauch zu machen
- Verbindung zur Maske über «Class» ProjektController



Demo Beispiel-Anwendung



«Form» Projekt

Zusammenfassung ChangeListener

- Bewusste Vermeidung von UI-Methoden und –Klassen
  => Verwendung durch Back-End-Klassen möglich
- Flexibles und erweiterbares Konzept
- Durch Vererbung nur Implementierung der individuellen Aspekte

Zusammenfassung

- Die Kapselung aller fachlichen Aspekte in einer Klasse erhöht die Wartbarkeit, weil der relevante Code sich an einer Stelle befindet
- Der gleiche Code kann sowohl im Front-End als auch im Back-End verwendet werden
- Durch die Ableitung von der «Class» BaseModel ist eine einheitliche Schnittstelle über alle Fachklassen gewährleistet
- Durch die Verwendung des MVC-Frameworks Konzentrierung auf die fachlichen Aspekte
- Das Framework ist leicht zu erweitern

Noch Fragen?

Kontakt & Download

Bernd Hort
assono GmbH
Lise-Meitner-Straße 1-7
D-24223 Raisdorf
Tel. +49 (0)4101/4 87 47
Mobil +49 (0)177/4 44 87 47
bhort@assono.de

Download der Folien & Beispiele
http://www.assono.de/blog.nsf/d6plinks/EntwicklerCamp2007

Folien Backup

Objektorientierung in LotusScript – Definition einer Klasse

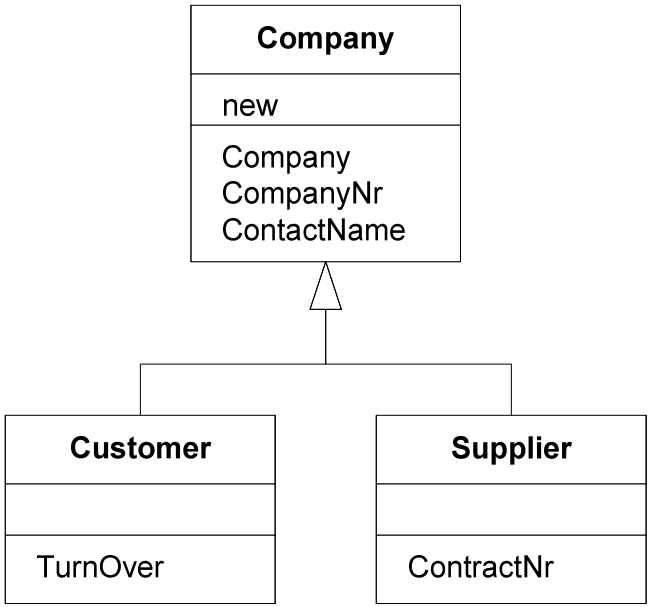
```
Class Company                                                          Klasse
        Private strCompanyName As String                               Membervariablen
        Private strCompanyNr As String
        Public ContactName As String

        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
```

Vererbung

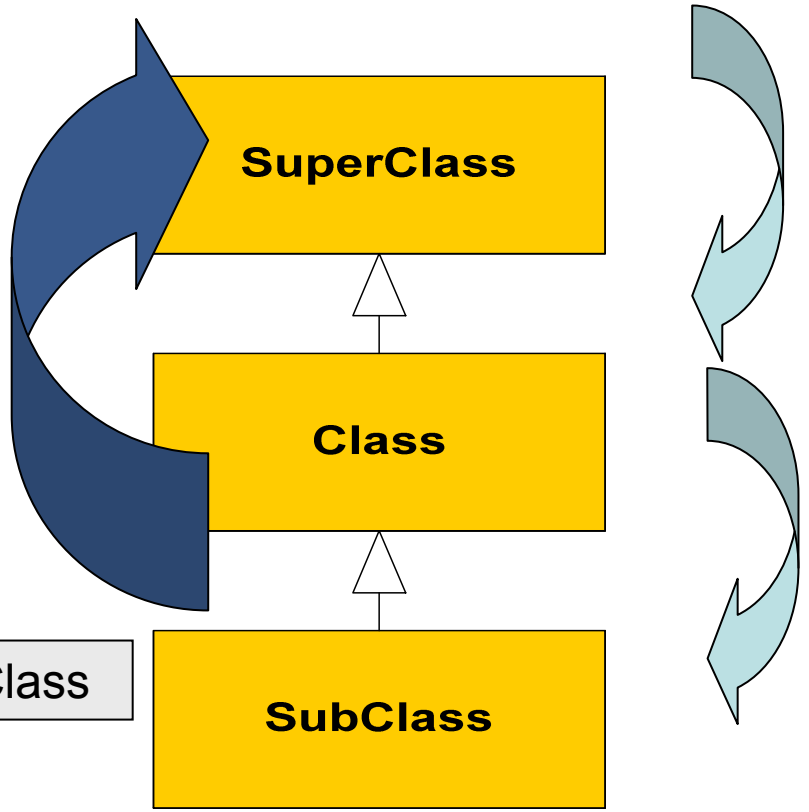
- Übernehmen von Properties und Methoden einer Klasse
- Ergänzung durch eigene Properties und Methoden




```
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

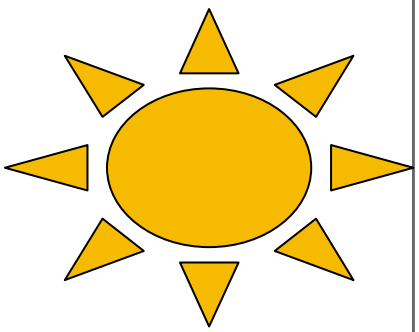
- Abgeleitete Klassen können einen von der Elternklasse abweichenden Konstruktur haben.

```
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

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!

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

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