assono

OOP in LotusScript – the next step

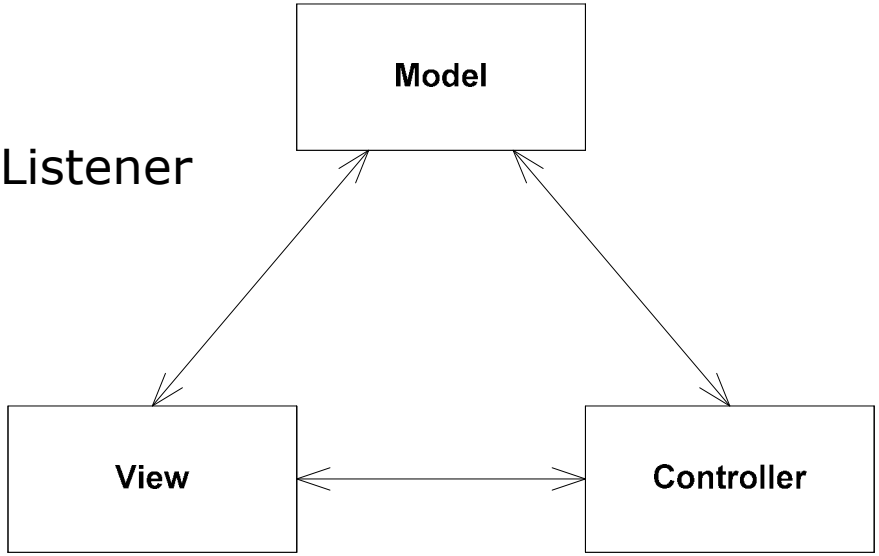
Building a MVC-Framework

English translation of a session from the EntwicklerCamp in Germany

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

Agenda

- Introduction
- Motivation
- Model View Controller Pattern
- «Class» BaseModel
- «Class» BaseController
- «Class» AbstractChangeListener
- «Class» ItemLevelHistoryChangeListener
- Relations
- Questions & Answers



Introduction

- Bernd Hort
- graduated in computer sciences (degree: Diplom-Informatiker)
- Lotus Notes application developer since 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

Why should we care?

No more overtime!

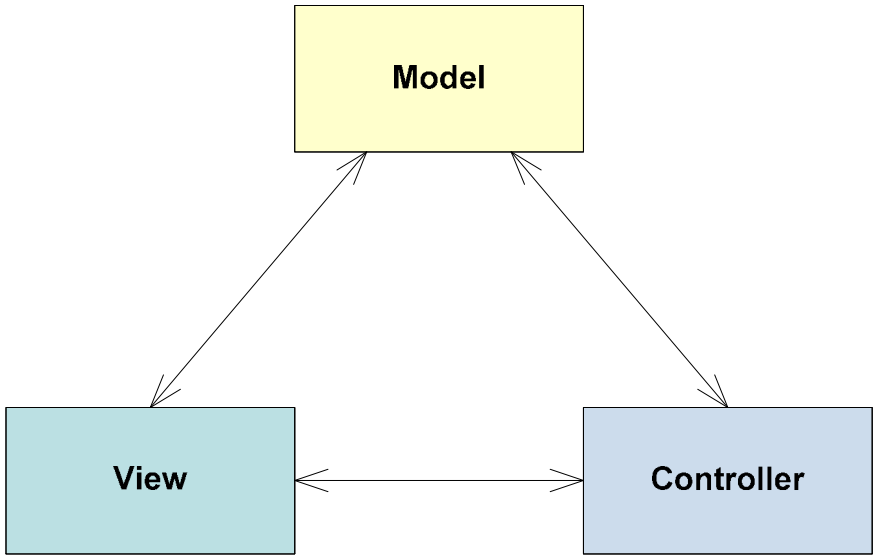
…and seriously?

- Development on a higher abstraction level
- Coping with complexity
- Higher level of reusability
- Less error-prone
- Easier to maintain

- Learn it now because it is used by nearly all OO-languages.

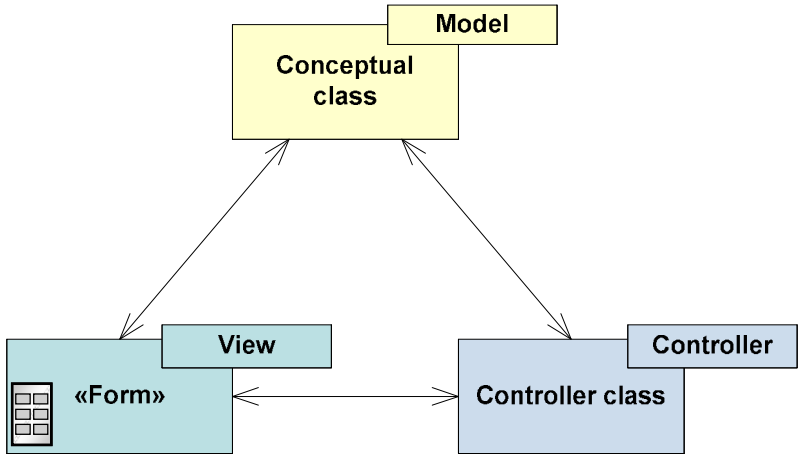
Model View Controller Pattern

- Originally introduced in Smalltalk
- Common OO-principle for the development of GUIs
- Separation of the domain classes from their presentation
  - *Model* – domain class
  - *View* – presentation
  - *Controller* – user interaction
- The domain class has no knowledge about their presentation!



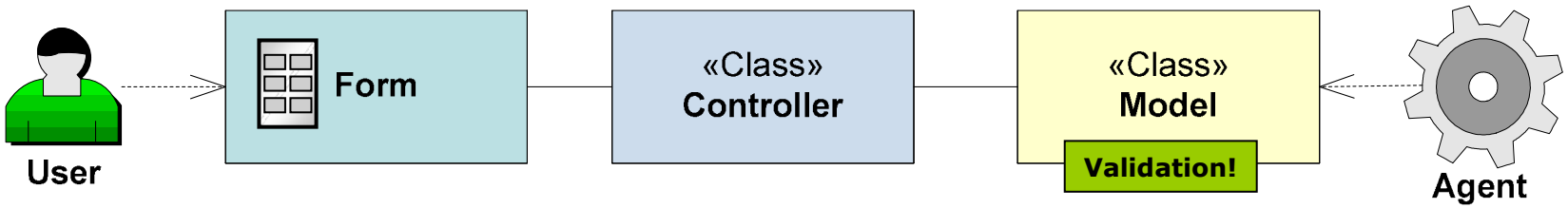
Model View Controller in Notes

- The form is used for the presentation of data and interacts with the user.
- All domain requirements are realized in a conceptual class.
- The controller class links the conceptual class to the form.



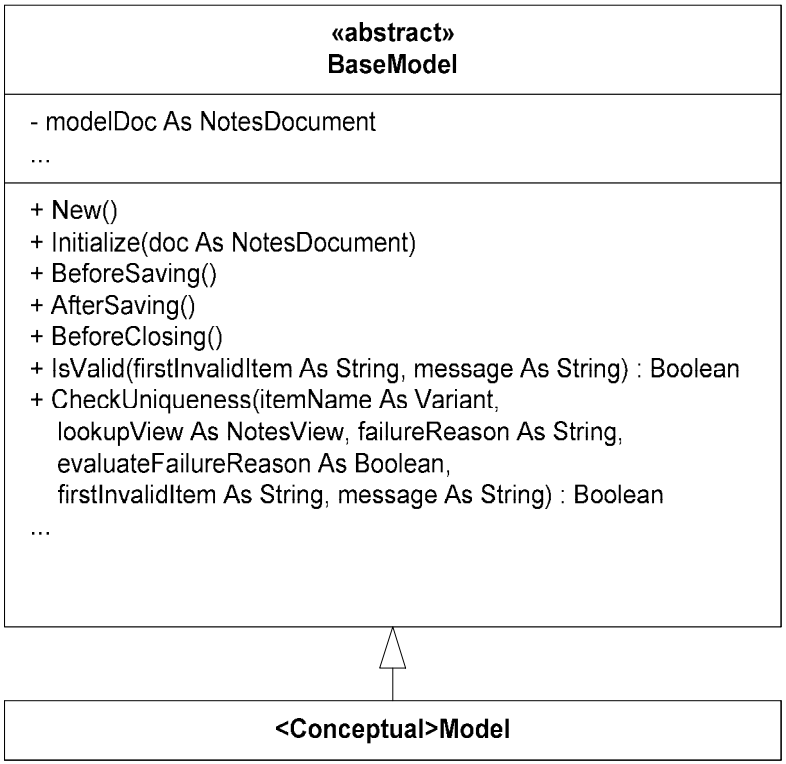
Realization of all the requirements in a conceptual class

- All the requirements from the domain model should be realized in a conceptual class.
- This includes in particular all input validations and plausibility checks.
- The conceptual class will use no UI-methods so it can be used by backend-agents.
- That means that the same method for checks and input validation will be used by agents and for input made by the user.



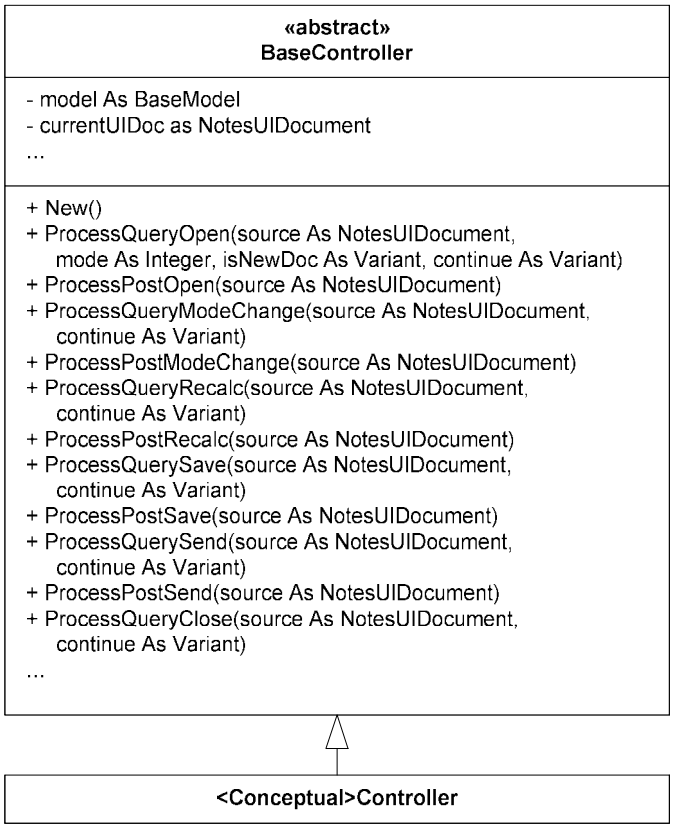
Abstract «Class» BaseModel

- Basis for all conceptual classes
- Most of the methods only have a signature
  i.e. the method has to be implemented in the sub classes
- No UI methods or classes allowed
- The conceptual class inherits from this class



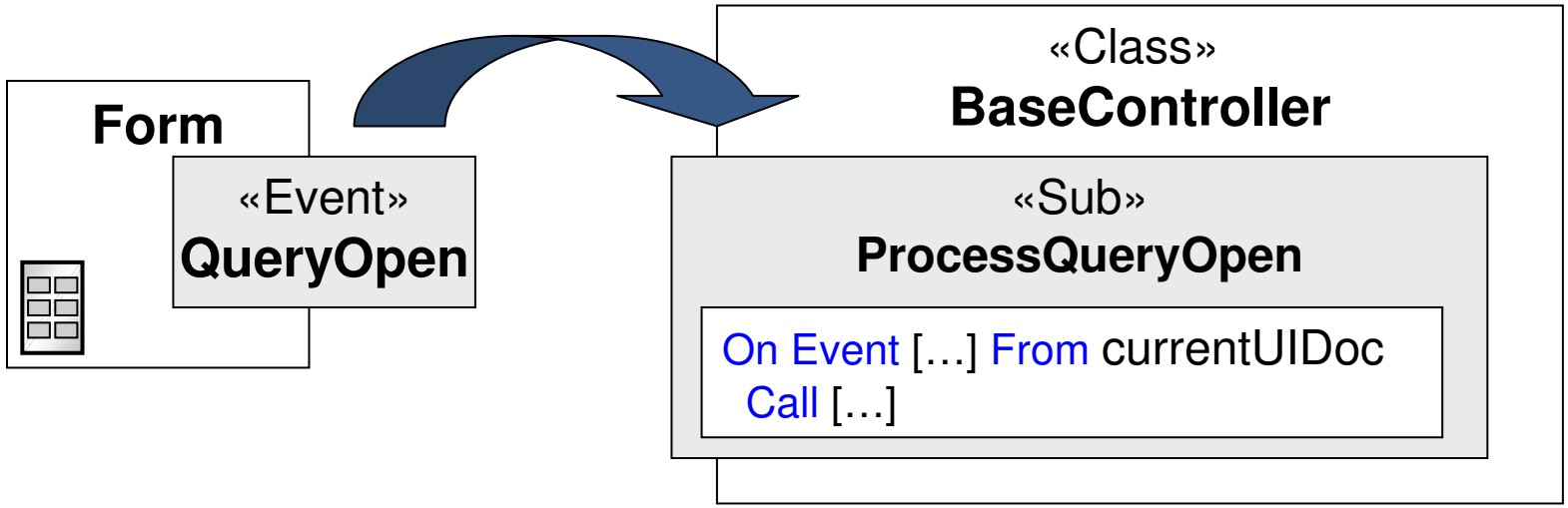
Abstrakte «Class» BaseController

- Connects the conceptual class with the form
- Handles all the events in the form



Form-Event-Handling in Controller Class

- The aim is to minimize the code in the form



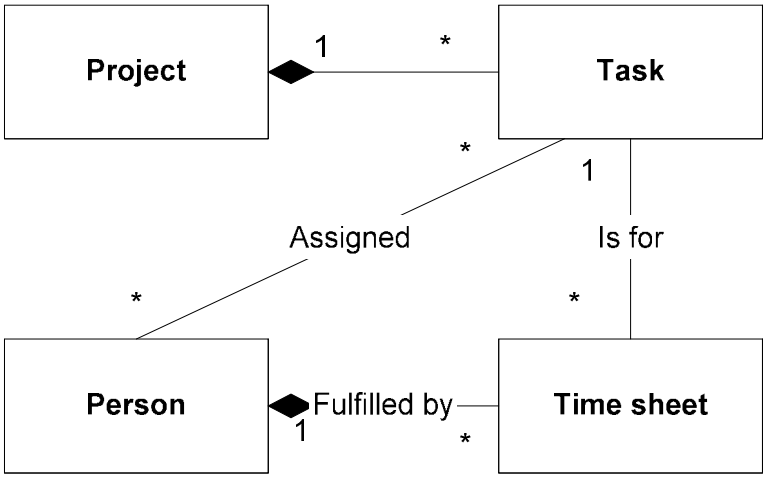

```
Sub Queryopen(Source As Notesuidocument, Mode As Integer, Isnewdoc As Variant,
Continue As Variant)
        Call CreateProjectController(source, mode, isNewDoc, continue)
End Sub
```

Mapping for 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
```

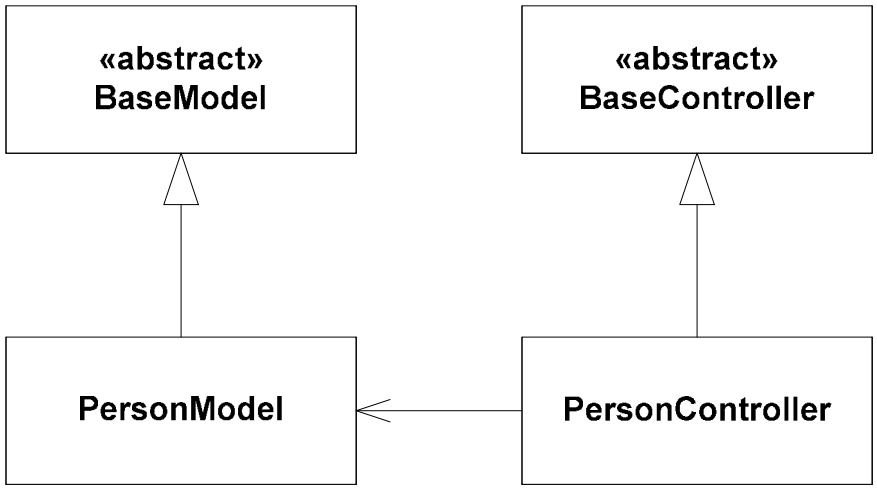
Sample application

- Simple project management
- Parts
  - Project
  - Task
  - Person
  - Time sheet



Person

- Conceptual requirements in the «Class» PersonModel
- Connects to the form via the «Class» PersonController

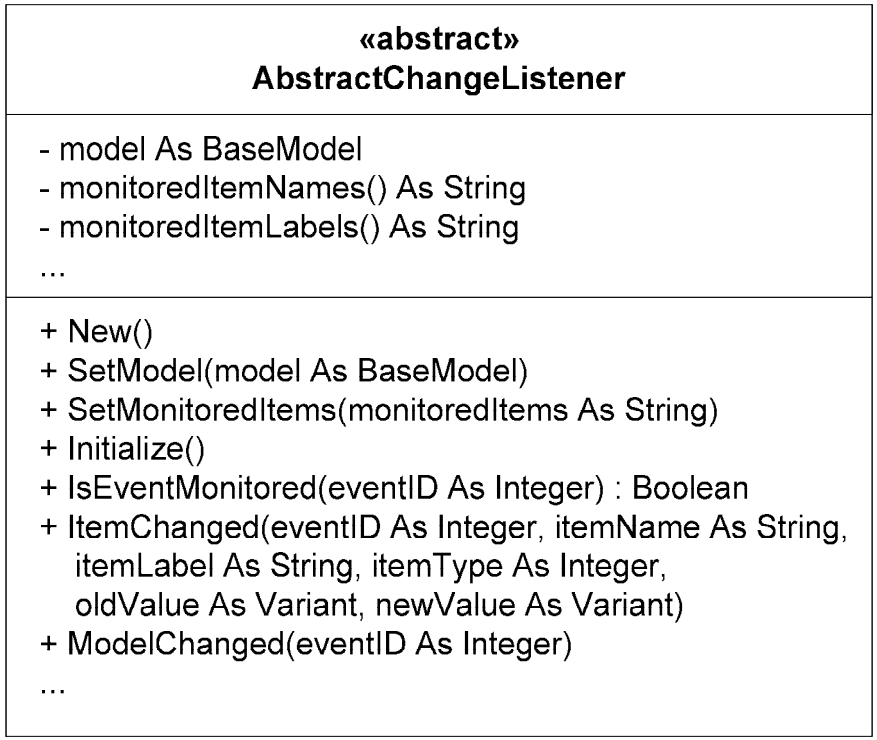


Demo Sample Application

«Form» Person

Monitoring of field value changes

- There are a lot of use cases where there is a need to monitor changes made to a field value.
  - History
  - Update of dependent documents
  - Notification if a threshold has been reached
  - …
- Instead of writing the same kind of code again and again it is a much better idea to encapsulate the functionality in a class

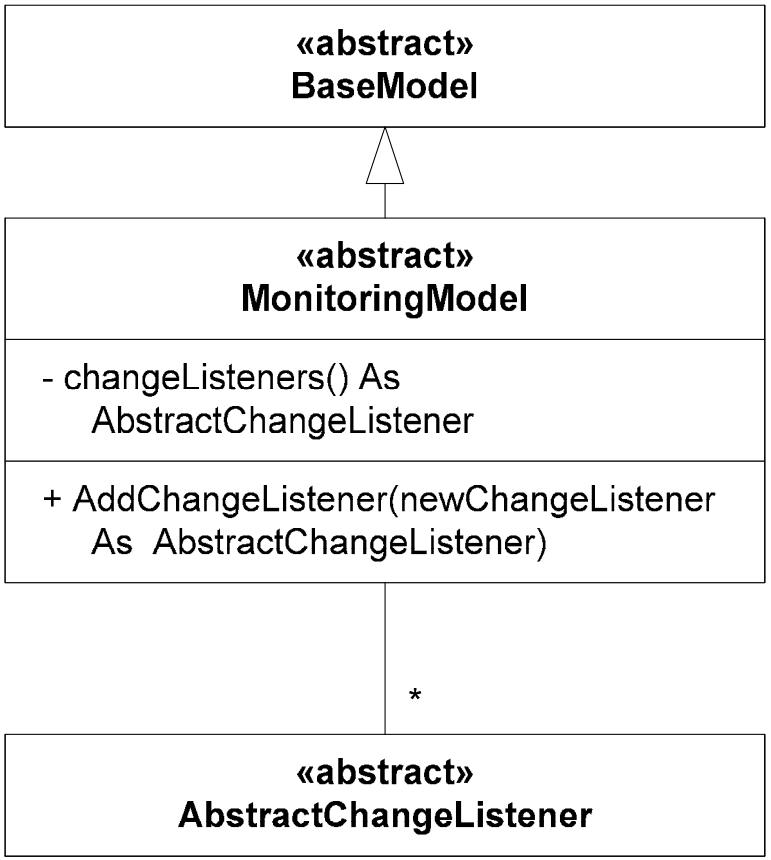


«Class» AbstractChangeListener – to monitor field level changes

- With the method
  SetMonitoredItems(monitoredItems As String)
  the fields to be monitored while be set
- The parameter is a string with a comma separated list of field names in the format
  Label|Field name
  or only the
  Field name
- e.g. "Person number | EmployeeID, Name, Last name|Lastname, First name|Firstname"

Inherited «Class» MonitoringModel

- «Class» MonitoringModel inherits from «Class» BaseModel
- Manages the interested ChangeListener [0..*]
- Holds internally a list of fields to be monitored
- Reacts on different events
- Informs the interested ChangeListener about changes on
  - Field level
  - Document level



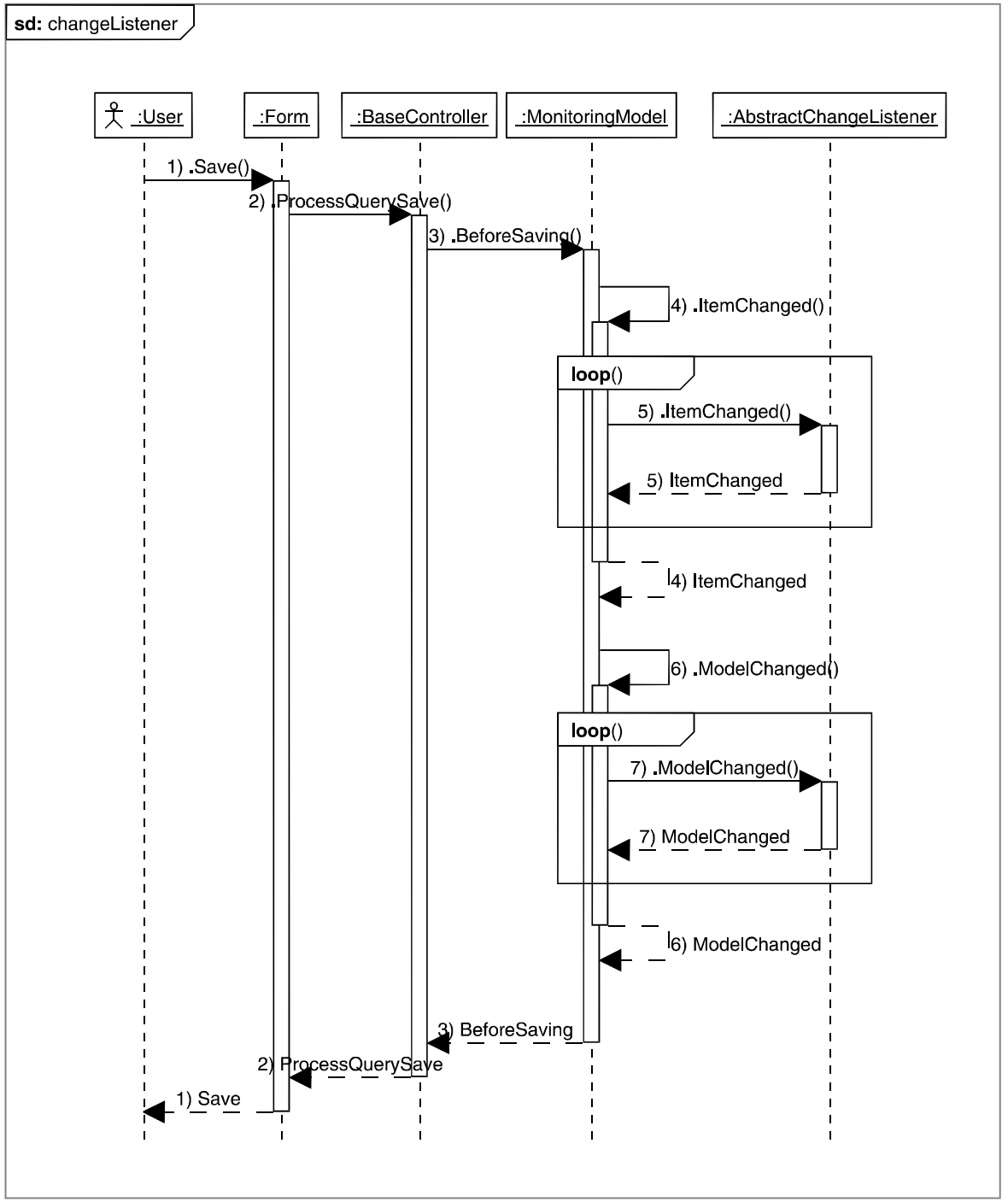
Possible Events

- The following events are supported

```
'/**************************************************************************
' * 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
```

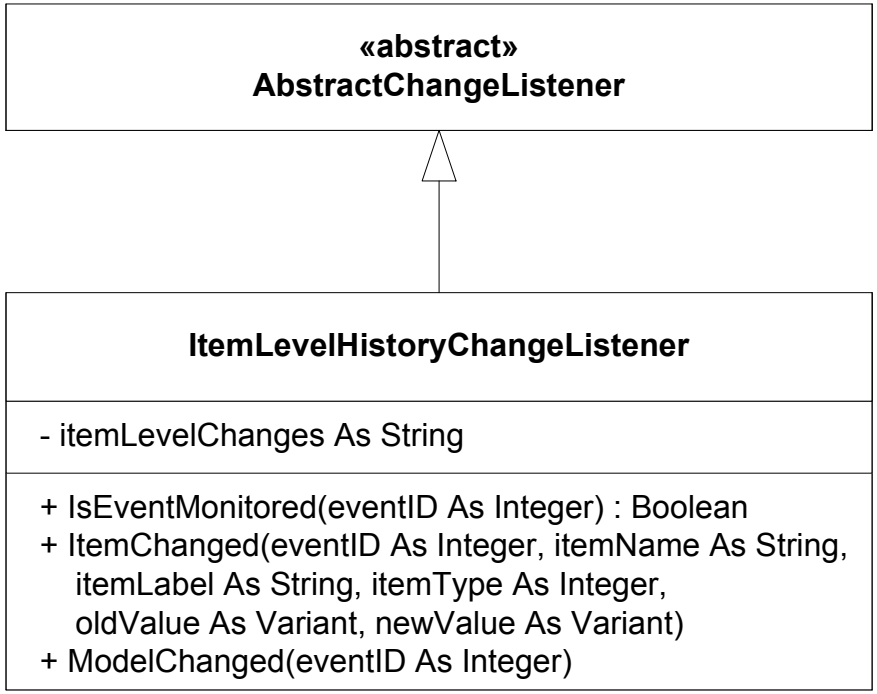
Sequence Diagram
ChangeListener

- Triggered by the event QuerySave
- Method MonitoringModel. BeforeSave()
- On a change at field level calls MonitoringModel. ItemChanged()
- Calls .ItemChanged() for all in this event interested ChangeListeners
- Calls MonitoringModel. ModelChanged()
- Calls .ModelChanged() for all the interested ChangeListeners



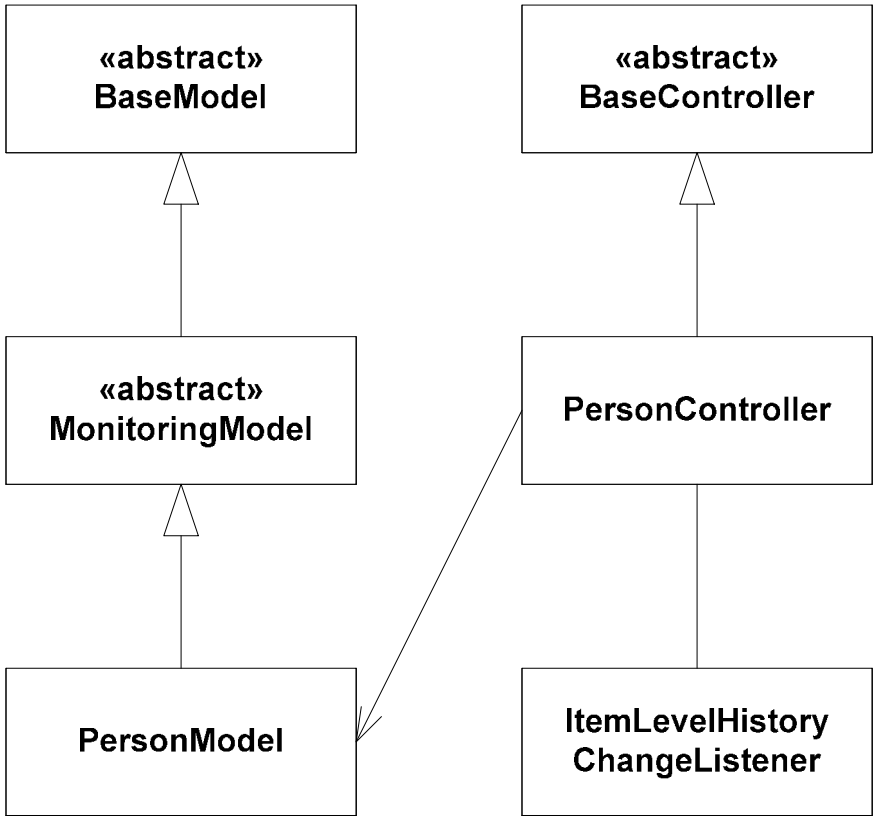
«Class» ItemLevelHistoryChangeListener

- Writes a history of changes at the field level
- Overloads the methods
  - IsEventMonitored
  - ItemChanged
  - ModelChanged



Person – Use of ItemLevelHistoryChangeListener

- Changing the superclass for the «Class» PersonModel into «Class» MonitoringModel
- In the «Class» PersonController the «Class» ItemLevelHistory-ChangeListener has to be defined



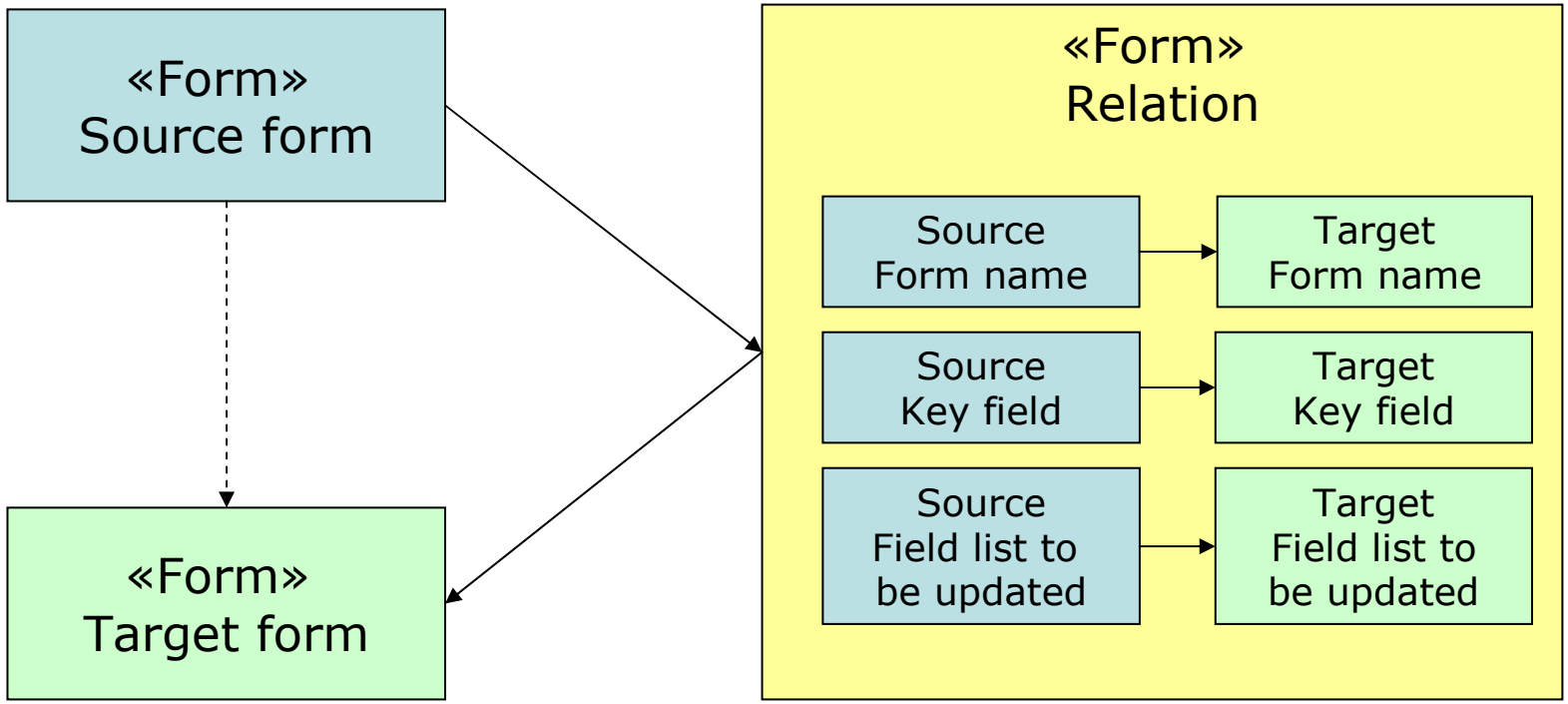
Demo Sample Application

«Form» Person

Relations

- The idea was inspired by a session held by Jens-B Augustiny at the EntwicklerCamp 2006.
- Makes updating dependent documents configurable
- Advantages:
  - Easy to enhance
  - Implicit documentation
  - „Dependency Injection“

Relations


«Form»
Source form
«Form»
Target form
«Form»
Relation
Source
Form name
Target
Form name
Source
Key field
Target
Key field
Source
Field list to
be updated
Target
Field list to
be updated

«Class» Relation

- Represents the relation document with the settings for source document
- Updates the dependent documents (target documents)

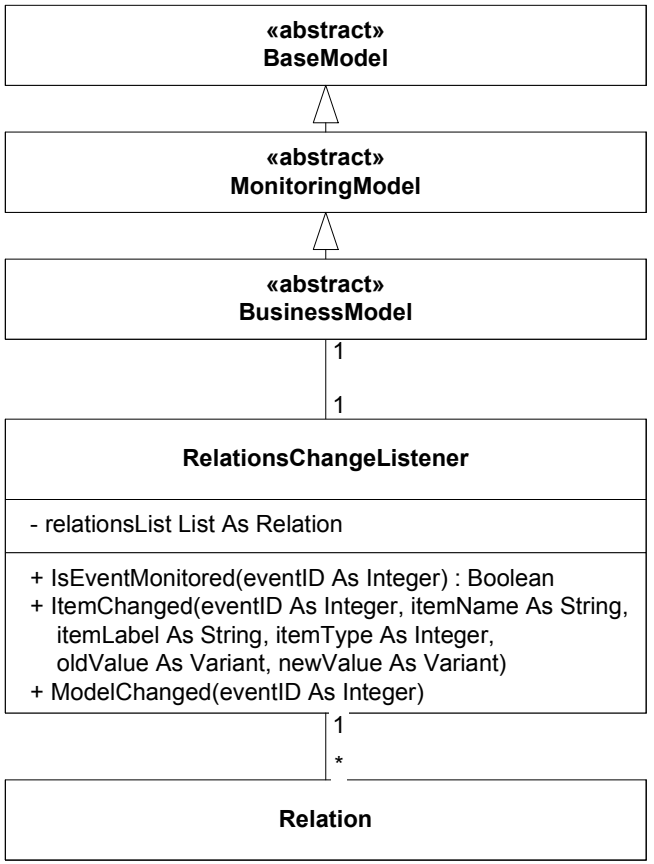
Relation

- 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)
...

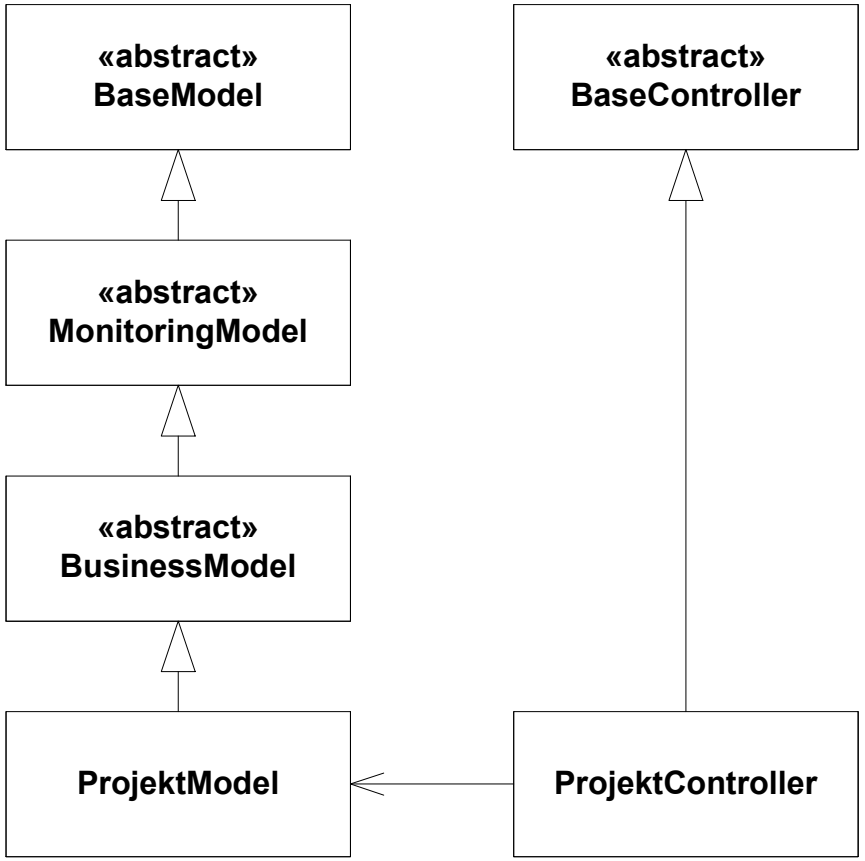
«Class» RelationChangeListener

- Manages all the for this form found relations
- Is been used by the «Class» BusinessModel



Project

- Conceptual requirements in the «Class» ProjektModel
- Is a subclass of the «Class» BusinessModel to take advantage of relations
- Connected to the form via the «Class» ProjektController



Demo Sample Application

«Form» Projekt

Summary ChangeListener

- Consciously avoiding the use of UI-methods and classes
  => so the change listener could be used by back-end classes
- Flexible and expandable concept
- Through inheritance only the individual aspects has to be implemented

General summary

- The encapsulation of all the conceptual requirements in a class makes the application a lot easier to maintain because all the relevant code is together in one place.
- The same code could be used in the UI and in the back-end.
- Through inheritance from the «Class» BaseModel a common interface is used for all conceptual classes.
- The use of the MVC-Framework helps the developer to concentrate on the conceptual part of the application
- The framework is easy to enhance.

Questions?

Contact & Download

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

Download of the slides and samples
http://www.assono.de/blog.nsf/d6plinks/EntwicklerCamp2007

Slides Backup

OO in LotusScript – Definition of a class

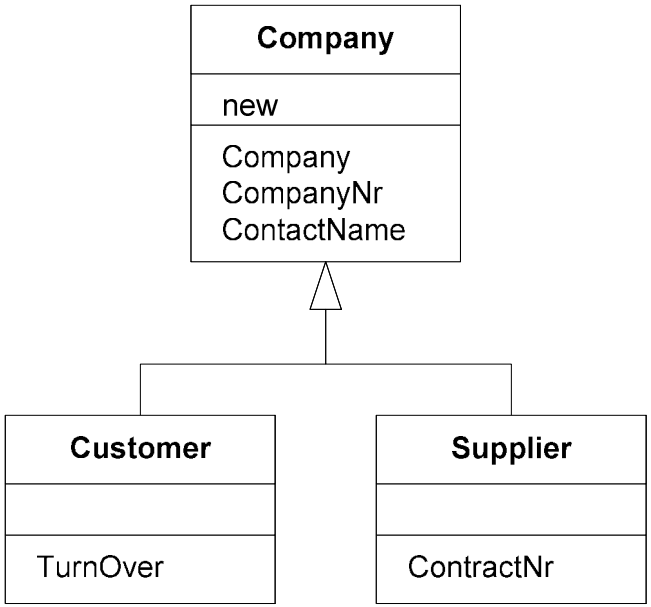
```
Class Company                                                    Class
        Private strCompanyName As String                         Member variables
        Private strCompanyNr As String
        Public ContactName As String

        Sub New (strCompanyName As String, strCompanyNr As String)
                Me.strCompanyName = strCompanyName
                Me.strCompanyNr = strCompanyNr                   Constructor
        End Sub 'Company.New

        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
```

Use of a class

```
Dim objCompany As Company 'Definition of the variable

Set objCompany = New Company ("DaimlerChrysler", "DC")
'Initialize Object
objCompany.ContactName = "Dr. Dieter Zetsche" 'Set of a public
member variable

objCompany.CompanyName = "DaimlerChrysler AG" 'Set of a
private member variable through a property

Messagebox objCompany.ContactName & " - " &
objCompany.CompanyName & " (" & objCompany.CompanyNr & ")",
64, "Demo" 'Use of the member variables
```

Inheritance

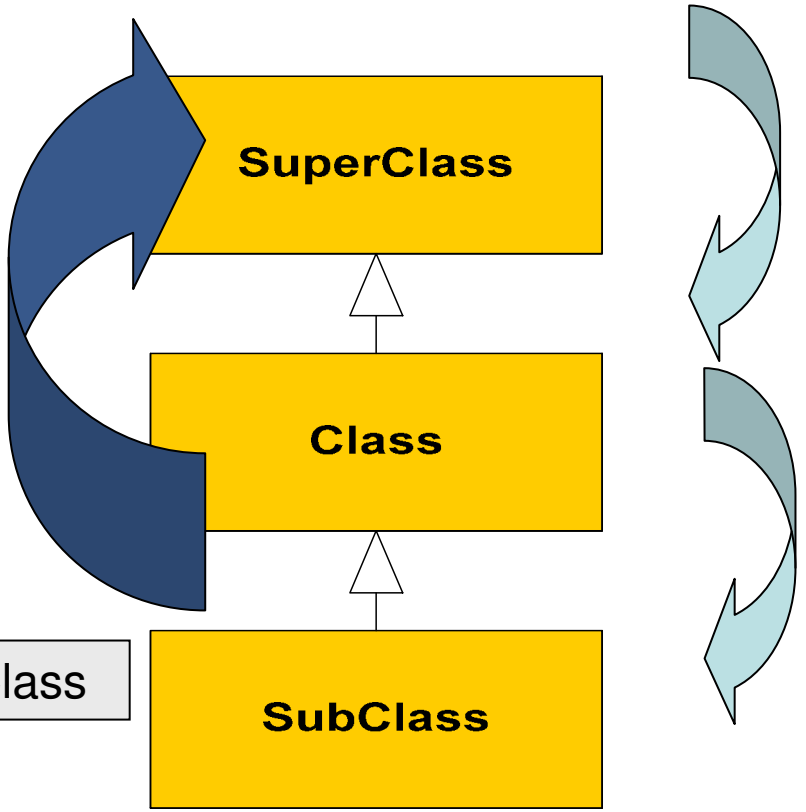
- Use of the properties and methods from the superclass
- Enhanced with own properties and methods




```
Class Customer As Company
        Public TurnOver As Double

        Sub New (strCompanyName As String, strCompanyNr As String)

        End Sub 'Customer.New

End Class 'Customer
```

Call stack for initialization of a sub class

- First always the constructor of the superclass is been called
- Followed by all other constructors
- Finally the constructor of the subclass is called




```
Set objSubClass = New SubClass
```

Overload of the constructor

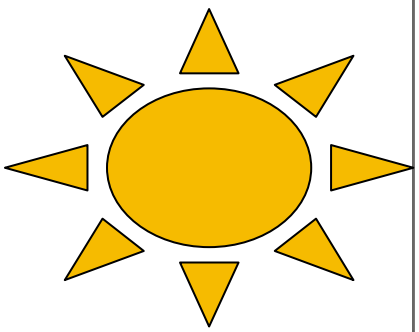
- Subclasses can have a different signature than the superclass.

```
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
```

Passing the parameters to the superclass

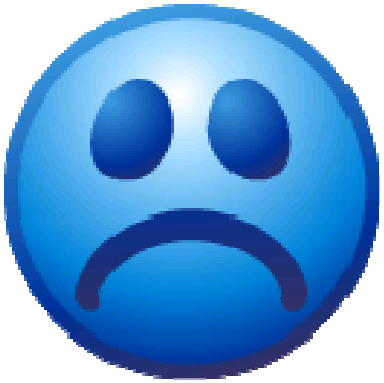
Overload methods and properties

- In the subclass methods and properties with the same name could be defined.
- The signature has to be identical with the superclass.
- To call methods and properties from the superclass use the following notation:
  Superclass..MethodName



Standard Notes-Classes could not be derived

- It is not possible to derive the standard Notes-Classes directly in LotusScript.
- The only way is to use the LSX-Toolkit.



Performance

- There is certain overhead with the use of classes
- While looping through a great amount of objects trigger the garbage collector with
  Delete Object
- If a lot of small Script Libraries has to be loaded for a form there might be a recognizable delay

➢ Dynamic loading of Script Libraries

Dynamic Loading of Script Libraries - References

- Bill Buchan – Lotusphere 2005
  - BP107 Best Practices for Object Oriented LotusScript
- IBM Redbook
  - "Performance Considerations for Domino Applications"
  - SG24-5602
  - Appendix B, Page 243
- Gary Devendorf Web Services Samples

„Factory"-Class to Create New Objects

- There has to be a "Factory" class next to the normal class
- The object variable has to be declared as a Variant

```
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)
```

„Dynamic Loading“ – The Magic

Using Execute together with a global variable

```
Public newObject As Variant 'Global defined

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 'The code in the string will be executed
        Set CreateClass = newObject.Produce(objErrorContainer)
End Function
```

Advantages & Disadvantages – „Dynamic Loading“

- Advantages
  - Script Libraries will only be loaded if they are needed.
  - Opening of a form is much faster
  - **During runtime it is possible to load different classes depending of the platform or the version!**

- Disadvantages
  - No more type checking from the compiler!

- Alternatives
  - See also OpenDOM on OpenNTF
    http://www.openntf.org/projects/pmt.nsf/ProjectLookup/OpenDOM

Tools – LotusScript.doc

- Generates a web-based documentation
- Similar to JavaDoc
- Supports additional comments
- http://www.lsdoc.org
- Freeware!

Tools - GhostTyper

- Inserting of code snippets directly form the Domino Designer
- http://www.ghosttyper.de
- Costs approx. 35,- €
- My GhostTyper-Archivs could be downloaded from http://www.hort-net.de/tools.html
- 5% discount if ordering through my website

Tools – Teamstudio Script Browser

- Shows all Subs, Functions and Classes in a database
- Analyses references
- http://www.teamstudio.com/support/scriptbrowser.html
- Freeware!
- More free tools via the blog of
  Craig Schumann / Chief-Developer from Teamstudio
  - http://blogs.teamstudio.com

Tools - Class Navigator from NotesHound

- Navigation within a the classes of a Script Library
- http://www.noteshound.com
- 49,95 $

  - Sorry, no discount!

Tools – Format LotusScript as HTML / RTF

- The for this slides used code coloring
- nsf tools
- http://www.nsftools.com/tips/NotesTips.htm#ls2html
- Freeware!
- Very interesting blog from Julian Robichaux
  - http://www.nsftools.com/blog