
Lotusphere®
2008

AD311

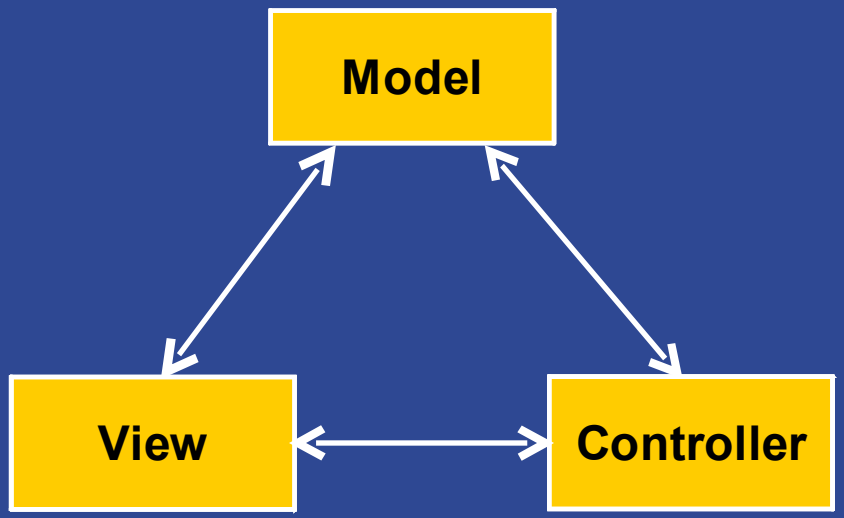
Object-Oriented Programming (OOP) with LotusScript – Take It To the Next Level

Thomas Bahn, Senior IT-Consultant, assono GmbH, Germany

Bernd Hort, Senior IT-Consultant, assono GmbH, Germany

Agenda

- Introduction
- Motivation
- OO Design Principles
- Sample Application
- Model View Controller Pattern
- Base Model
- Base Controller
- Change Listener
- Time to wrap it up



Introduction

Who we are

Introduction

- Bernd Hort, 37
- Graduated in computer sciences (degree: Diplom-Informatiker)
- Lotus Notes application developer since 1995
- IBM Certified Application Developer (R4 - 7)
- IBM Certified System Administrator (R4 - 7)
- IBM Certified Instructor SA & AD (R5 - 7)

Introduction (cont.)

- Thomas Bahn, 37
- Graduated in mathematics (degree: Diplom-Mathematiker)
- Cofounder of assono GmbH, an IT consulting company in Germany
- Developing in Java and RBDMS (OCP Admin) since 1997
- IBM Lotus Notes and Domino professional since 1999
  - Development, Administration, Consulting, and Education
- IBM Advanced Certified Application Development and System Administration (R4 – 7)
- Focus on interfaces to other systems (e.g., RDBMS, SAP R/3), Domino Web applications, and GUI programming

Motivation

Object-Oriented Programming (OOP) – Why should you care?

Motivation

- So – why **should** you care?

No more overtime!

Motivation (cont.)

- … and seriously?
  - Develop on a higher abstraction level
  - Cope with complexity - divide and conquer problems
  - Increase flexibility and extensibility of your solutions
  - Improve the reusability of your code
  - Reduce errors in your code
  - Enhance your codes' readability and maintainability
  - Accelerate the development process
- Learn it now because it is used by nearly all modern languages.

OO Design Principles

Object-Oriented Programming is more than just writing classes

OO Design Principles

- KISS – Keep It Simple, Stupid
  - Avoid unnecessary complexity, prefer simple solutions
- Principle of Least Astonishment
  - Astonishing solutions are most often hard to understand
- DRY – Don't Repeat Yourself
  - Avoid code duplication
- Separation of Concerns Principle
  - Every class should have only one responsibility

OO Design Principles (cont.)

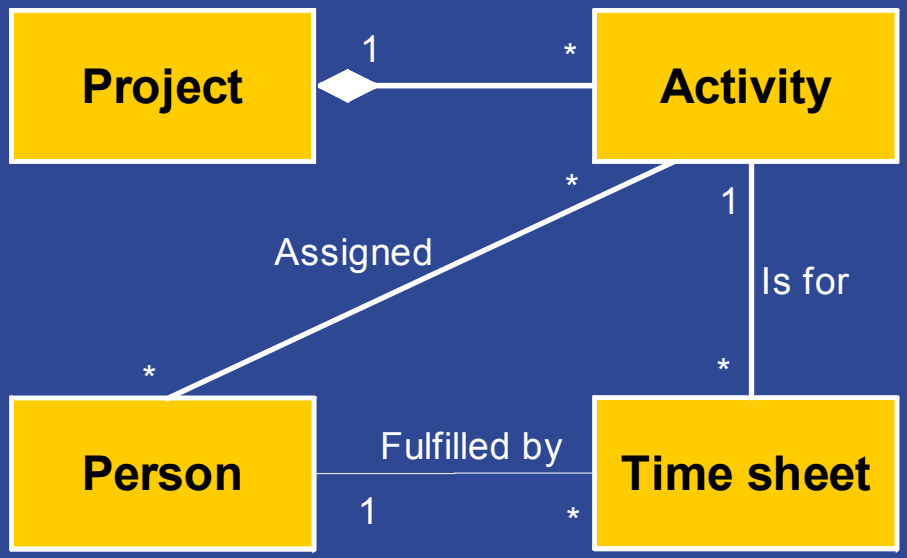
- Open-Closed Principle
  - Classes should be open for extensions, but closed for modifications
- Encapsulate What Varies
  - Changes don't affect stable portions of your code
- Favor Composition over Inheritance
  - So you can change behavior at run-time, not compile-time
- Keep the Public Interface Lean
  - Design as private as possible

A Sample Application

Beginning with the result

A Sample Application

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



Demo

The Sample Application

Lotus software

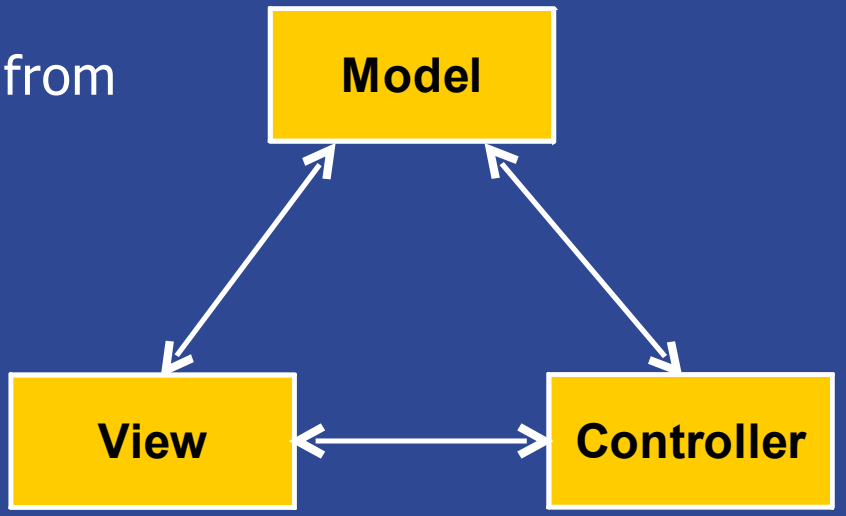
Lotusphere 2008

Model View Controller Pattern

Separation of Concerns at the highest level

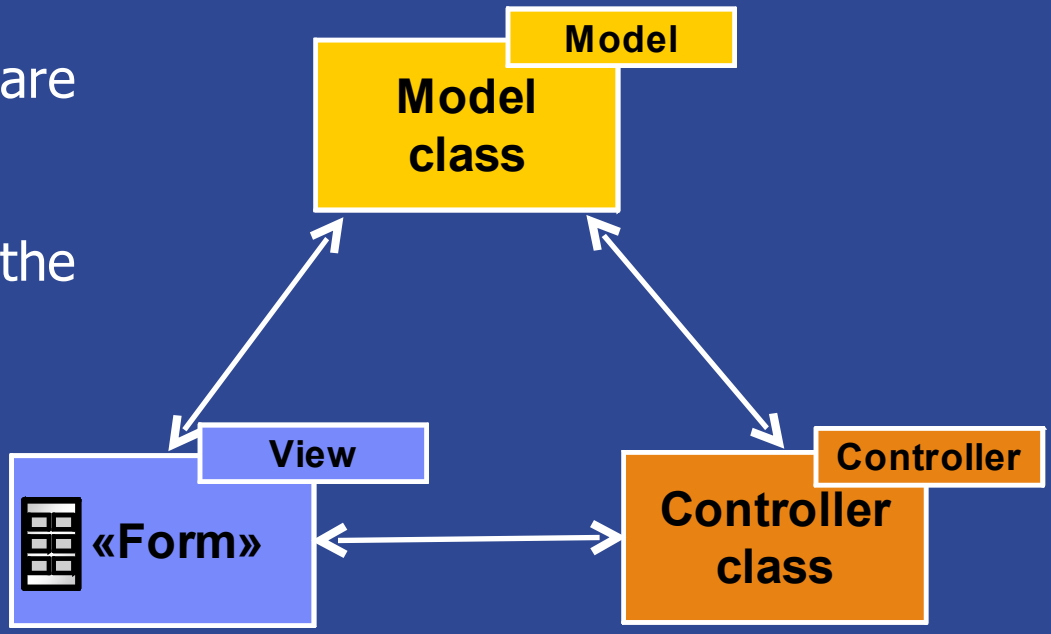
Model View Controller Pattern

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



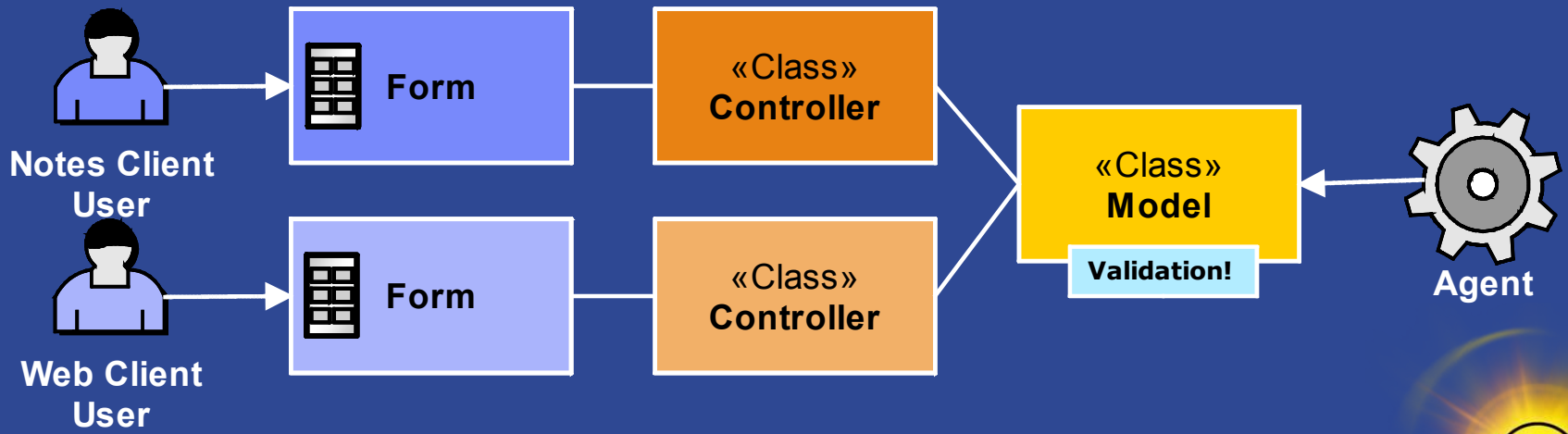
Model View Controller in Lotus Notes

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



Use Same Validation Code For All Clients And Agents

- All the requirements from the problem domain should be realized in the model class
  - This includes in particular all input validations and plausibility checks
- The model class will use no UI-methods so it can be used independently from the presentation (e. g. by backend-agents)
- That means that the same method for checks and input validation will be used by all clients: Notes, Web, and agents



Demo

«Form» Person

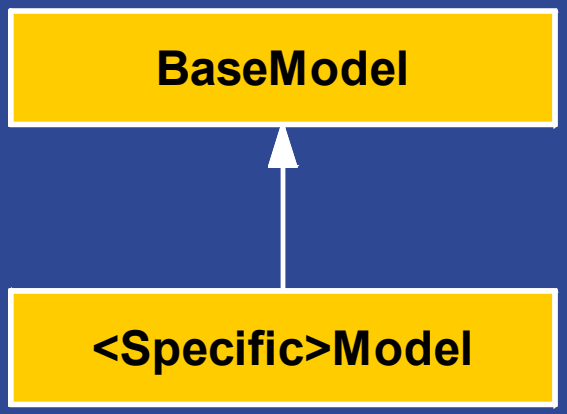
(User's perspective – input validation)

«Class» BaseModel

Only care about your business – the «Class» BaseModel will do the rest

«Class» BaseModel

- Base for all model 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
- All specific model classes inherits from this class

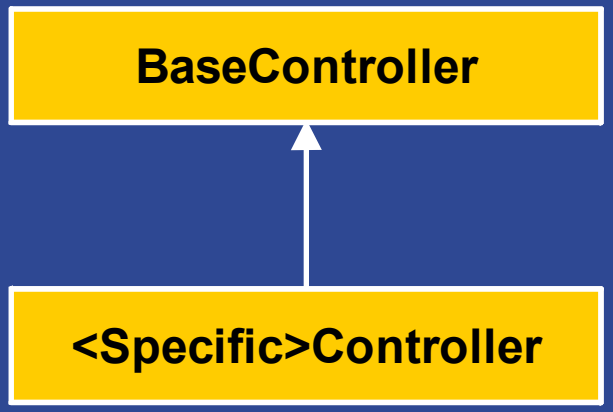


«Class» BaseController

The glue between the model and the form

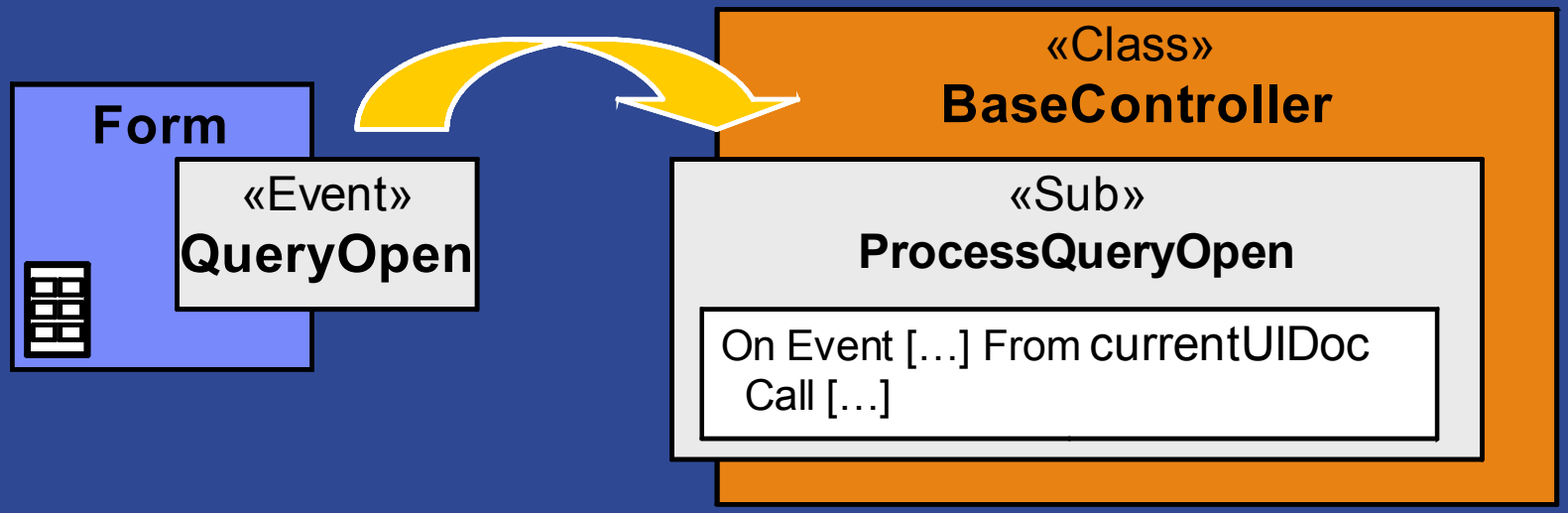
«Class» BaseController

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



Delegate Form Event Handling to Controller Class

- The aim is to minimize the code in the form



Mapping Form Event Handling

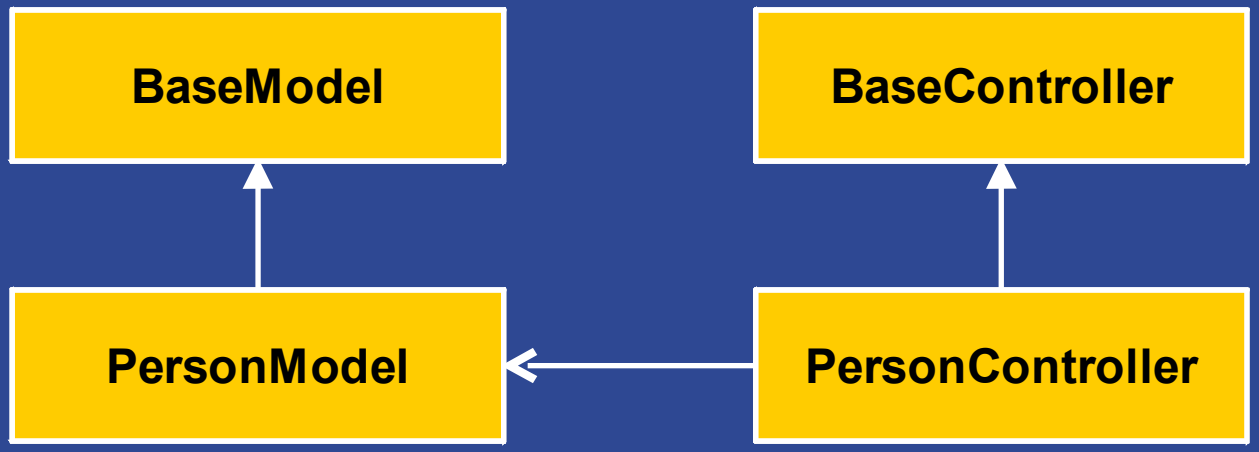
```
Public Sub ProcessQueryOpen(source As NotesUIDocument, mode
  As Integer, isNewDoc As Variant, continue As Variant)

  [...]
        ' register all event handlers
        On Event PostOpen From source Call processPostOpen
        [...]
        On Event QuerySave From source Call processQuerySave
        [...]
        On Event QueryClose From source Call processQueryClose
  [...]

End Sub ' BaseController.ProcessQueryOpen
```

Person

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



Demo

«Form» Person

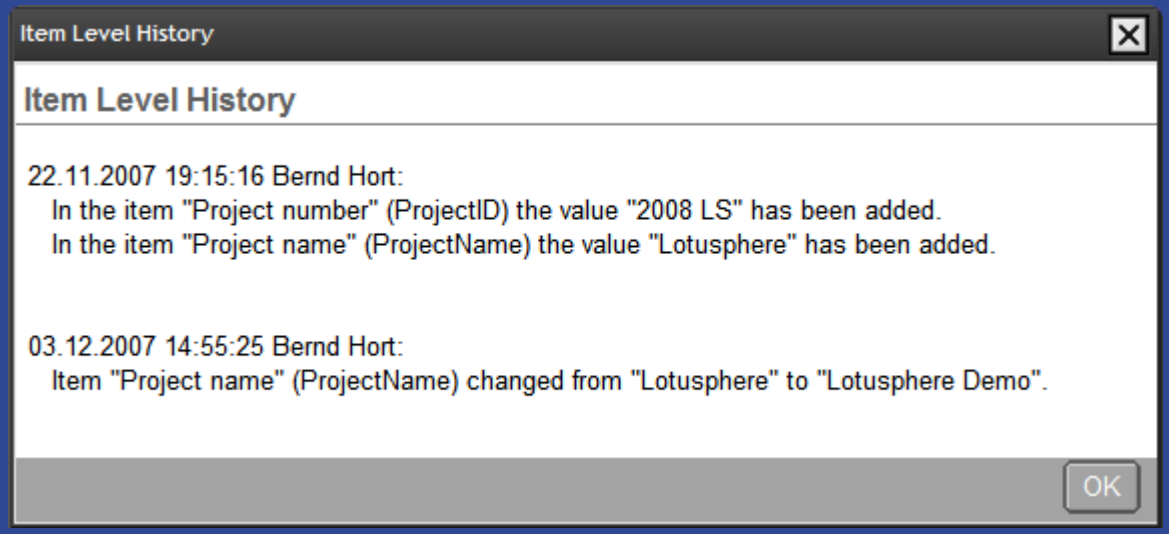
(Source Code Review)

Change Listener

Let the framework handle changes – a flexible and extendible approach

Item Level Change History

- Business problem: for compliance reasons, changes to all editable fields in some forms should be logged:
  - Who has done the change?
  - When has the change been done?
  - Which field has been changed?
  - What were the field's contents before and after the modification?



Demo

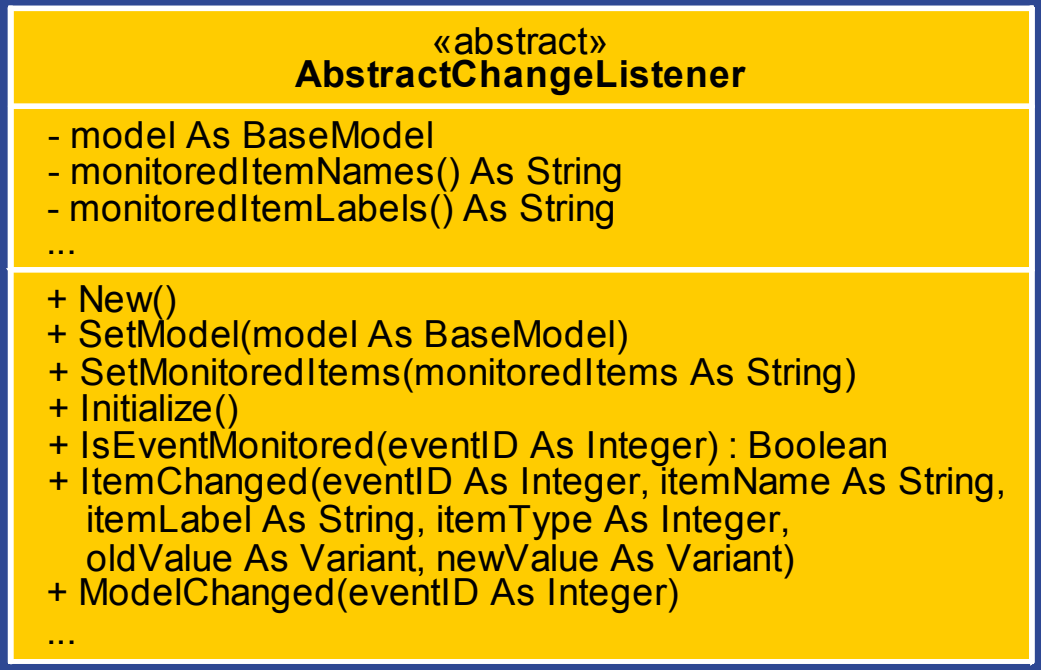
The field level history from a user's perspective

Lotus software

Lotusphere 2008

Monitoring of Field Value Changes

- A lot of use cases require to monitor field modifications
  - History and logging
  - Update of dependent documents
  - Reaction on exceeding a limit
  - …
- Instead of repeatedly writing the same kind of code, encapsulate the responsibility in a class (or two)



Our Solution: ChangeListener Classes

- In our framework we use the ChangeListener classes, which inherit almost everything necessary from AbstactChangeListener
- A concrete ChangeListener subclass only defines
  - which fields to monitor (cf. SetMonitoredItems)
  - when to react (before or after saving, before closing)
  - what to do for each changed field
  - what to do once for a changed document
- This gives us all the flexibility and extensibility we need!

The Puzzle's Second Part: MonitoringModel

- To enable more than one ChangeListener concurrently, we need a central "registry" – the MonitoringModel
- MonitoringModel is a subclass of BaseModel, which…

1. administrates a list of ChangeListeners and
2. gathers and administrates **one** list of items, at least one registered ChangeListener is interested in – the monitored items
3. is responsible for taking snapshots of all monitored items' values
   - when a document has been loaded
   - after a document has been saved

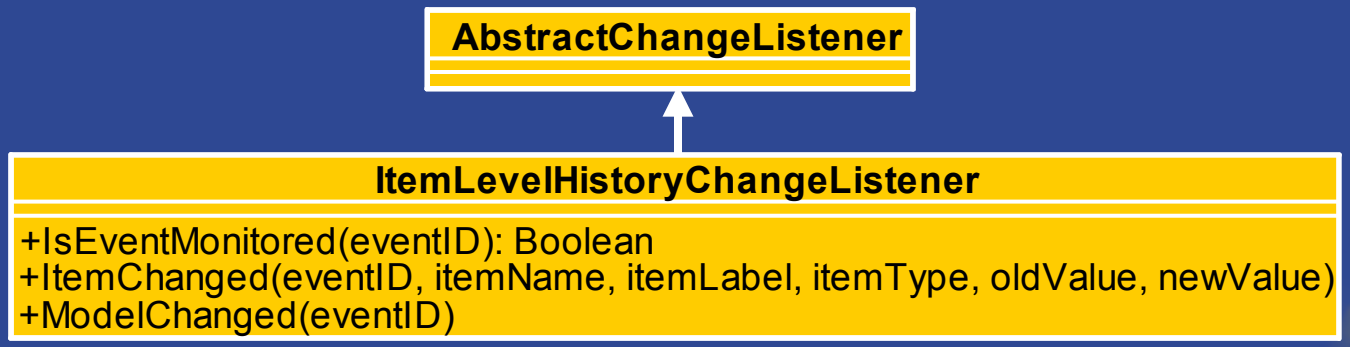
The Puzzle's Second Part: MonitoringModel (cont.)

- MonitoringModel is a subclass of BaseModel, which… (cont.)

4. compares item values to snapshots
   - before saving,
   - after saving and
   - before closing of the document
5. notifies registered ChangeListeners, when an item, they are interested in, has been changed
6. notifies registered ChangeListeners that the document has been changed (when at least one monitored item has been changed)

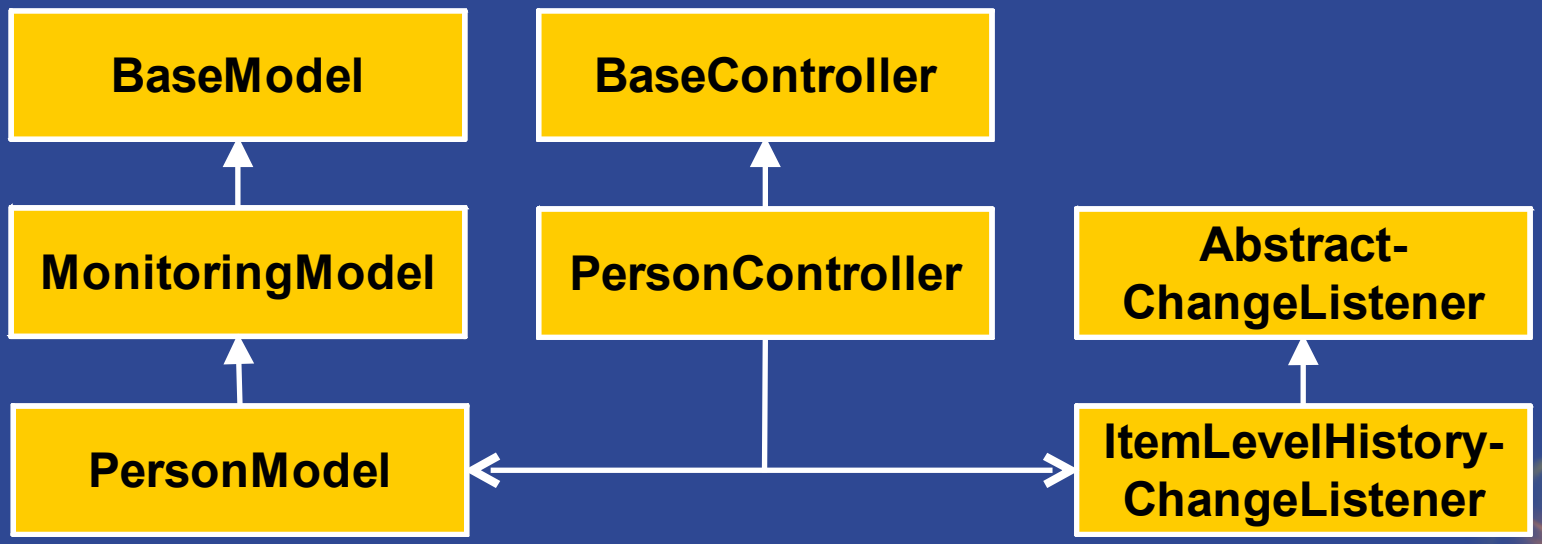
Back to Business

- How can these two classes be used to solve our problem?
- First, we create the ItemLevelHistoryChangeListener subclass of AbstractChangeListener
- This class only overwrite these three methods
  - IsEventMonitored: only react on BEFORE_SAVING events
  - ItemChanged: assemble a list of item changes (which item, old and new value)
  - ModelChanged: prepend the current user and timestamp to the list of item changes and append the string to the log item in the document



Usage of ItemLevelHistoryChangeListener

- Change the superclass for the «Class» PersonModel to «Class» MonitoringModel
- Create in the constructor of «Class» PersonController an instance of «Class» ItemLevelHistoryChangeListener and register it with the PersonModel
- And we're done!



Demo

How PersonController.New() creates and registers a FieldLevelHistoryChangeListener

(Source Code Review)

Time to wrap it up

What we have tried to tell you

Lotus software

Lotusphere 2008

Time to wrap it up

- Accelerate your application development using OOP
  - Simplify code reuse through inheritance and delegation, avoid code duplication
  - Extend functionality by subclassing or delegation, don't modify tested and time-proven code anymore
  - This minimizes the risk of errors and therefore the time for testing and debugging
- Increase maintainability of your applications
  - Reduce complexity by assigning one and only one responsibility to each class
  - This distribution of responsibilities makes your code easier to read and understand
- Lighten your job and use the MVC pattern
  - and other OO design patterns as well
- Take heed of the OO design principles
- Continue to learn more about OO programming, design principles and patterns

Resources

- assono Blog
  - www.assono.de/blog
- LotusScript.doc
  - JavaDoc style documentation of LotusScript code
  - www.lsdoc.org
- Class Navigator
  - Noteshound
  - www.noteshound.com
- Teamstudio Script Browser
  - www.teamstudio.com/support/scriptbrowser.html

Q & A

See us in the Speaker's room:
Toucan 2 (here in the Swan hotel)

or send an email to tbahn@assono.de
or bhort@assono.de

Lotus software

Lotusphere 2008

Evaluation form

Don't forget your evaluation form!

Lotus software

Lotusphere 2008

The workshops and sessions offered have been prepared by IBM or the session speakers and reflect their own views. They are provided for general information purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participants. Such information is provided "as is" without warranty of any kind, and IBM assumes no liability or responsibility for the content of any information made available during the workshops and sessions, including without limitation information you obtain during general or individual discussions and/or question and answer sessions.

Material in this directory (including without limitation any advertisements) regarding third parties is based on information obtained from such parties and their Web sites. No effort has been made to independently verify the accuracy of the information. Mention or reference to all non-IBM products is for informational purposes only. Information is provided "as is" without warranty of any kind. This document does not constitute an express or implied recommendation or endorsement by IBM of any third party, its product or service.

This directory is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or in any way related to, this directory and/or any information herein. Nothing herein shall create any warranties or representations from IBM or its licensors, or alter the terms and conditions of the applicable license or agreements governing the use of IBM software or services. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability. In addition, certain information may be based on IBM's current product plans and strategy which are subject to change by IBM without notice.

Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Lotus software

Lotusphere 2008