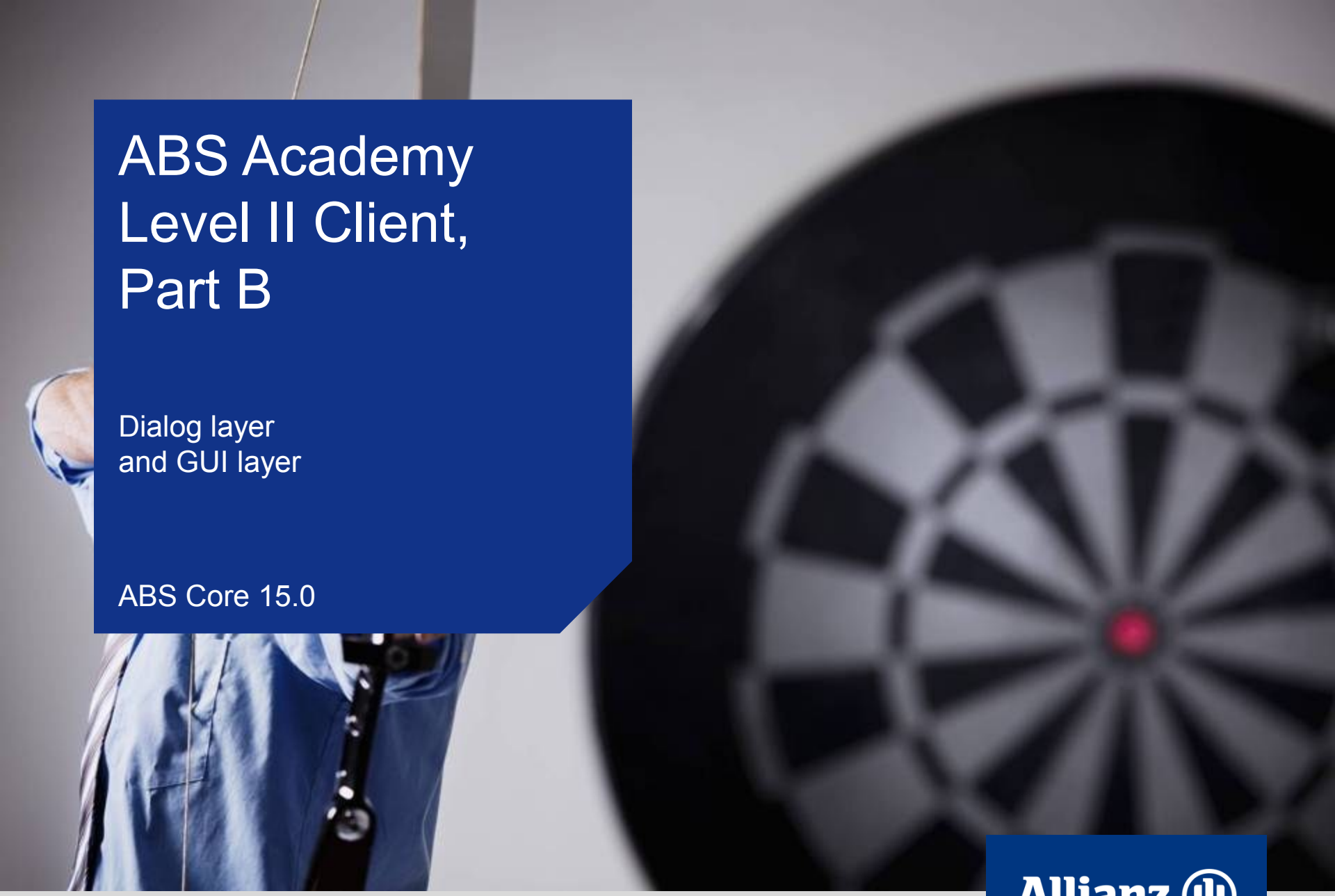


ABS Academy Level II Client, Part B

Dialog layer
and GUI layer

ABS Core 15.0

Allianz

Workshop overview

	ABS Academy Level II Client	
Prerequisites	Part A	**Part B**
Basic technologies	Persistence and business layer	**Dialog and GUI layer (rich client)**
▪ *Advanced Java* ▪ *Eclipse IDE* ▪ *JUnit* ▪ *Eclipse concepts and plug-in development* ▪ *SWT/JFace*	▪ Introduction, history and architecture overview ▪ Basic concepts (core view) - Persistence layer, DataContainer - Business logic layer ▪ Extension concepts (customer view)	▪ *Basic concepts (core view)* - *Dialog logic* - *GUI* - *Internationalisation* ▪ *Extension concepts (customer view)* ▪ *Cross-functional topics*
	Practice, practice, practice …	

Agenda

Part B

1 Basic concepts (core development)
- 1.1 Dialog layer
- 1.2 GUI layer
- 1.3 Internationalisation

2 Extension procedure (customer-specific adjustments)
- 2.1 Dialog layer
- 2.2 GUI layer
- 2.3 Internationalisation

3 Cross-functional topics

Allianz

Allianz

The A3k dialog layer

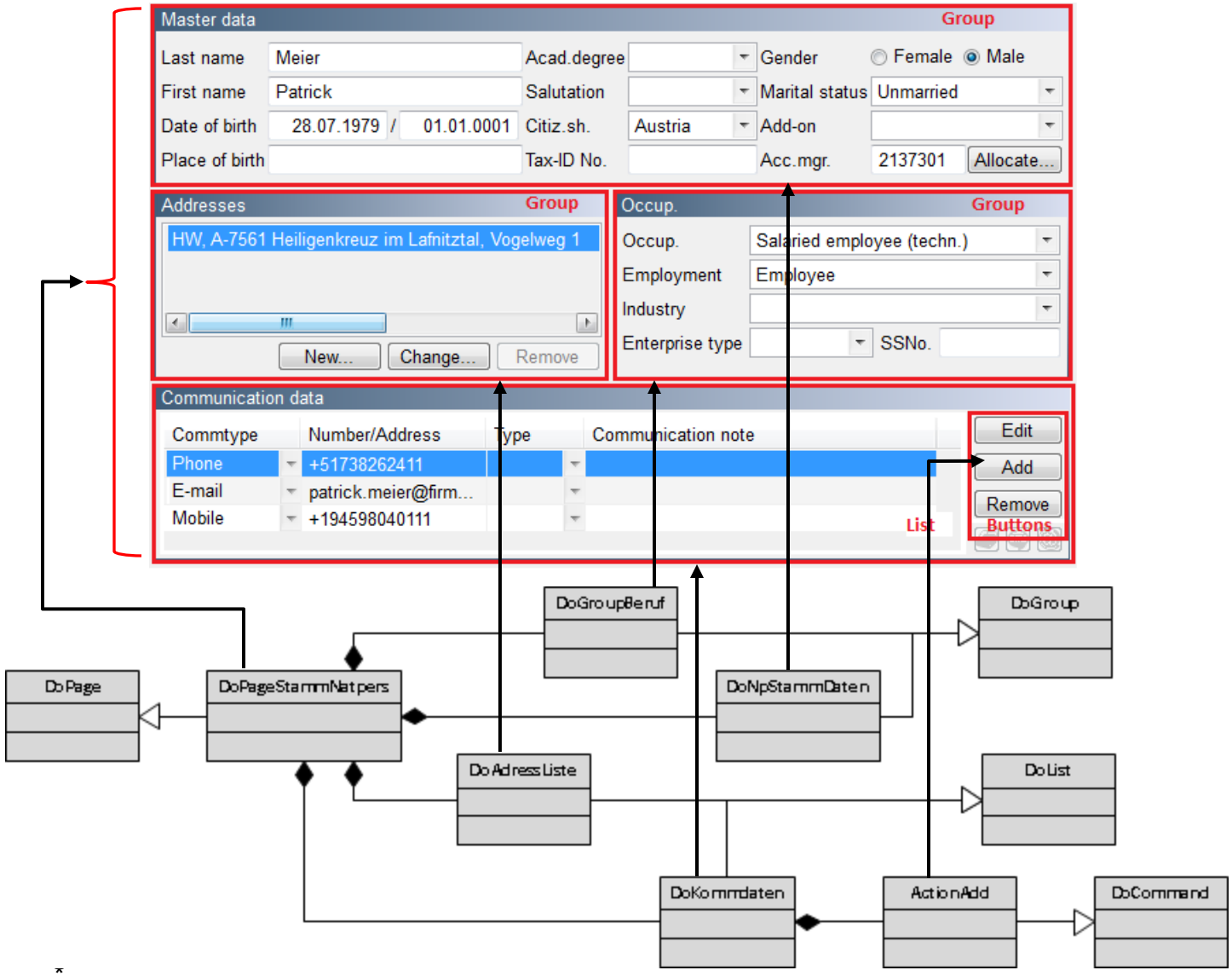
Responsibilities

- Business logic façade
- Groups business logic data for display/processing
- Hierarchical structure in windows, pages, groups, input fields…
- Abstracts the GUI control elements
- Each GUI element has a corresponding dialog object

Clarification of terms

- A3k dialog layer is not a dialog layer in the conventional sense of the term
- Corresponds to a View Model in the design pattern Model/View/View Model

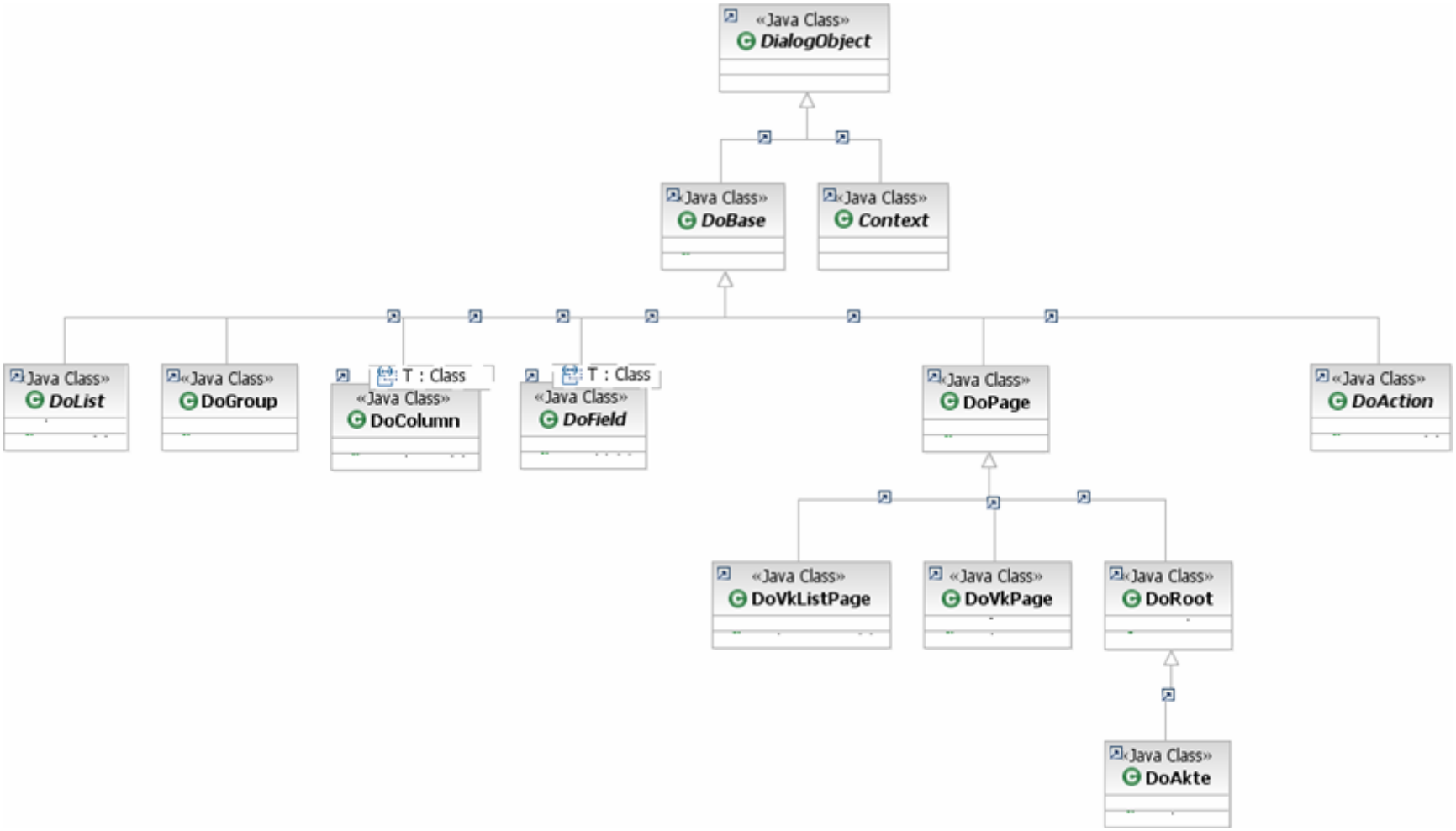
Example – Dialog object structure



Dialog layer classes

Dialog layer class	Corresponding GUI control elements
DoAkte	File, screen
DoVkPage	VKS (usage class view) page header
DoPage	Page, pop-up window
DoGroup	Grouping
DoList	Table, list
DoRelation<T>	Table, list
DoField<T>	Text field, drop-down list box, checkbox, radio button
DoFieldAttribute<T>	Special DoField for connection to BoAttribute
DoColumn<T>	Table, list column
DoColumnAttribute<T, V>	Table, list column
DoCommand, DoAction	Button, menu entry

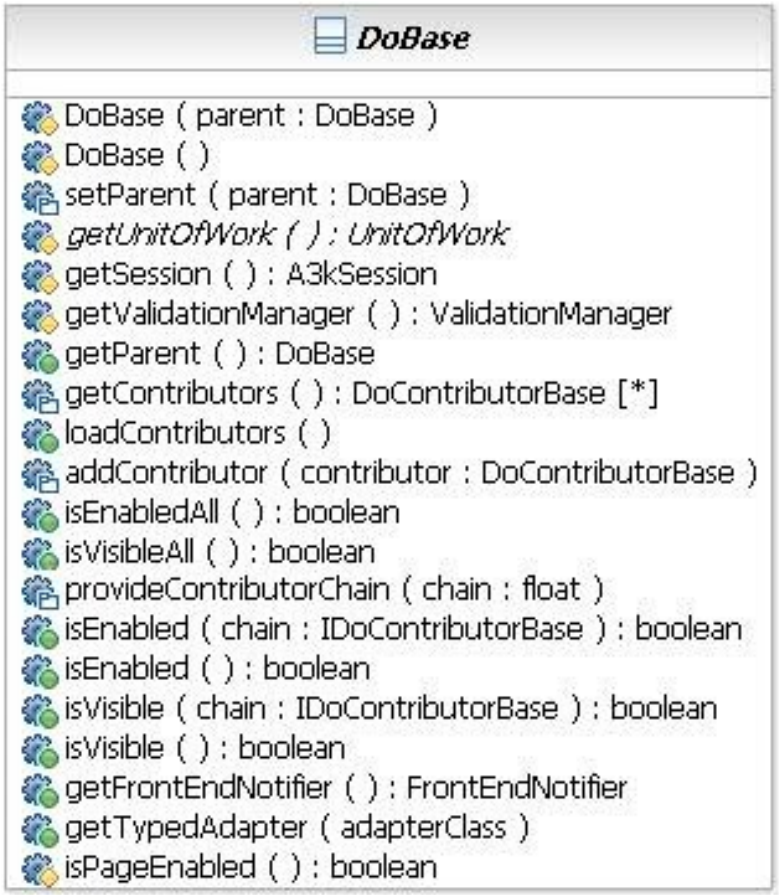
Framework dialog class hierarchy



DoBase – Dialog object base class

Common features of all dialog objects:

- Hierarchy structure:
  Reference to the parent element
  (constructor parameter parent)
- Visibility status: `isVisible`
- Editability: `isEnabled`
- Access to
  - `UnitOfWork`
  - `A3kSession`
  - `ValidationManager`
  - `FrontEndNotifier` (legacy concept)
- Administration of registered contributors
  (see chapter on extensions)



DoPage – Page class

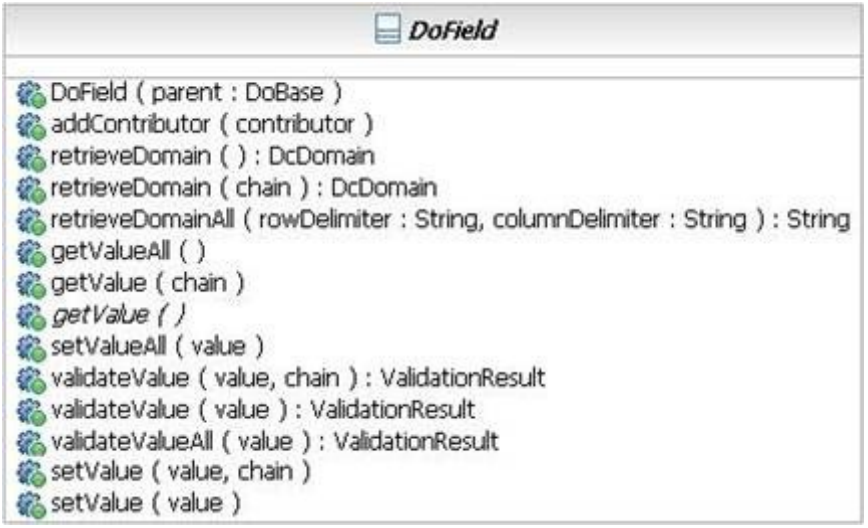
Responsibilities of derived classes:

- Aggregate and instantiate subordinated dialog objects (`constructor` or `setContext`) and offer them via getters
- Method `setContext:`
  - Obtains the context object from the framework, e. g. with the `DbKey` of the current persistent object
  - Has to pass this context and any changes on to sub-elements
  - Should contain the code that obtains the required business objects from the context object and, where appropriate, initializes dialog objects using these
- Method `validate`: Page validation

DoField – Input field, drop-down list box, checkbox or radio button

Implementation of derived classes as an inner class of the superordinated `DoGroup, DoList, DoPage`

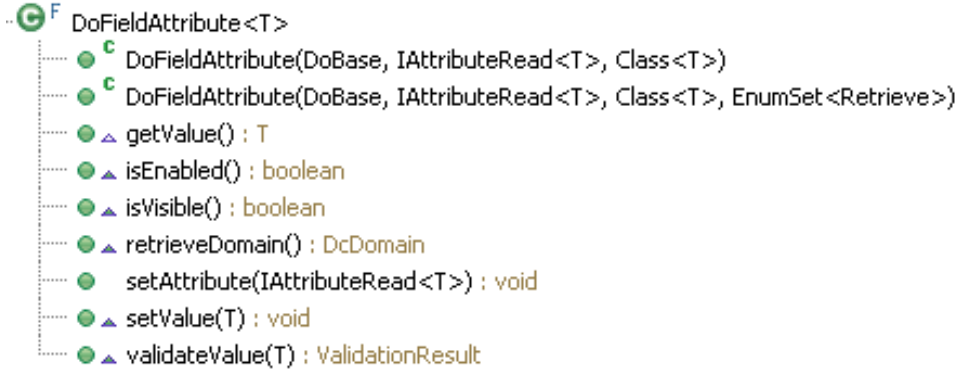
- Always to be implemented: `getValue`
- For fields that can be edited: `setValue`, where appropriate for field validation `validateValue`
- For domain values: `retrieveDomain`
- Base class itself does not contain any data storage, generally delegation to business object
- Base class specifically for link to `BoAttribute`: `DoFieldAttribute` (see next slide)



DoFieldAttribute – DoField for BoAttribute

Special `DoField` for connecting to a `BoAttribute`

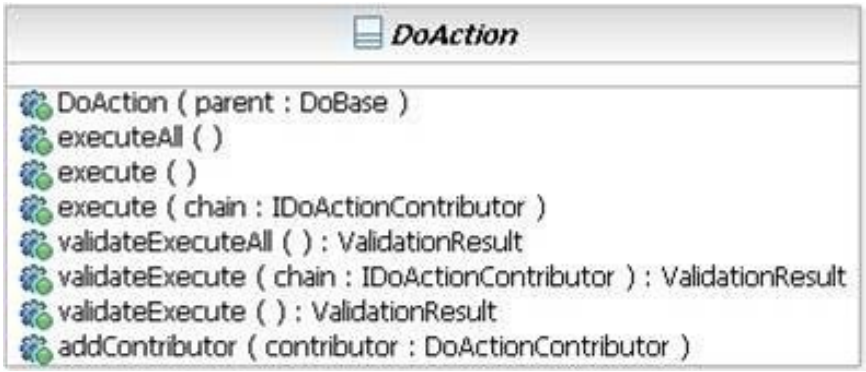
- Gets the data from a `BoAttribute`
- A set of retrieve modes (`EnumSet<Retrieve>`) can be passed to define what should be automatically refreshed after a value is set (see ‘Updating the GUI’)



DoCommand, DoAction – Button, menu entry

Implementation as the internal class of the superordinated component `DoGroup, DoList, DoPage`

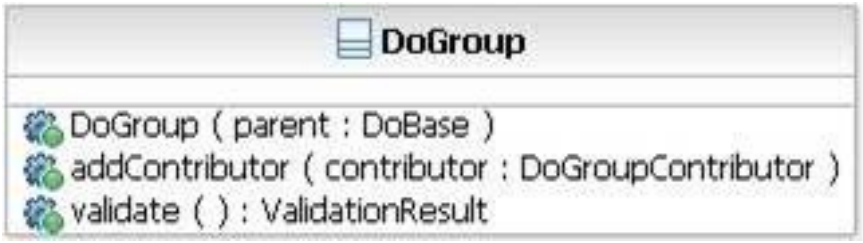
- `DoCommand`:
  - Contains only `isVisible, isEnabled`
  - Command definition has to be set in the constructor: call `setDefinition`
- `DoAction`:
  - Not for new development, only for older pages with PowerBuilder emulation GUI
  - `Method execute`: event handling after button or menu item gets clicked
  - `Method validateExecute`: performs validations before execute where appropriate



DoGroup – Group

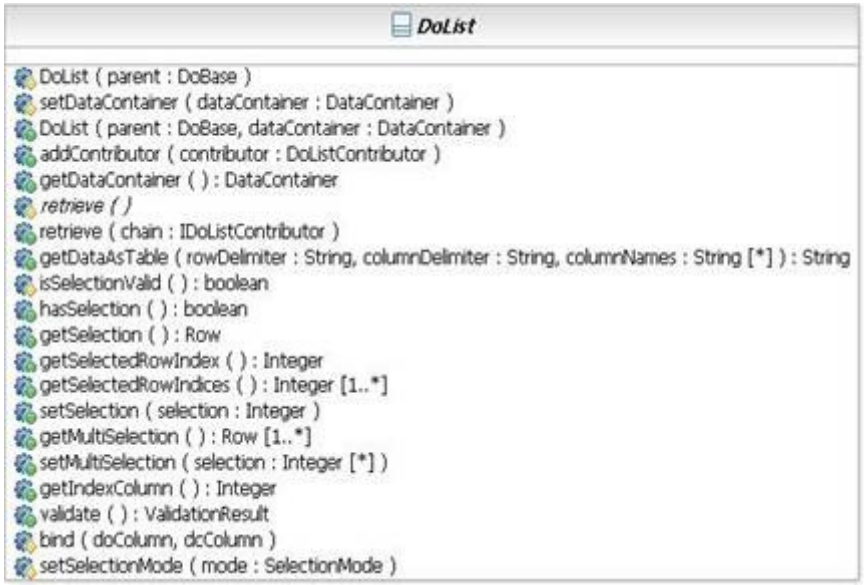
Container for a group of dialog objects

- Grouping idea
- Contains a number of `DoFields` and `DoCommands/DoActions`
- Aggregates these subordinated dialog objects:
  - Instantiate (in constructor or `setContext`)
  - Keep in instance variables
  - Offer via getters



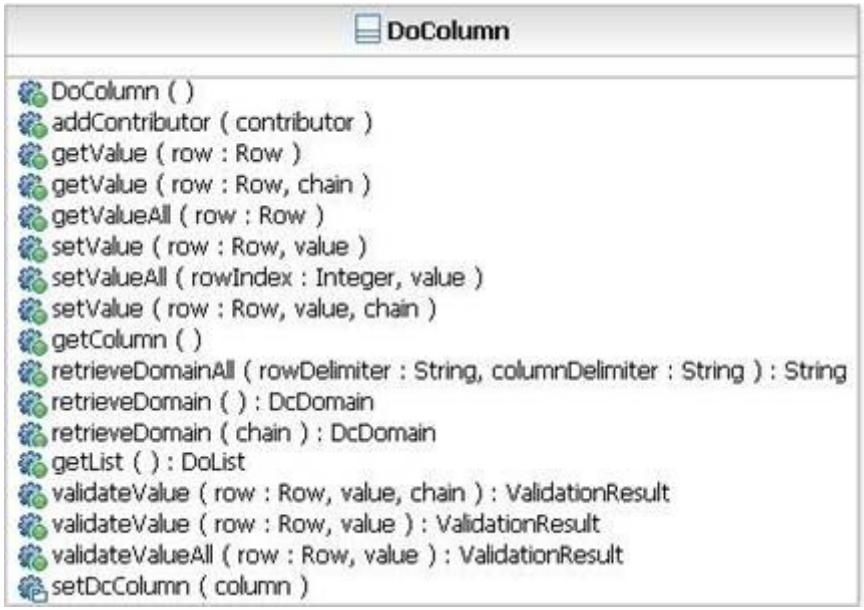
DoList – Table, list

- Linked to a `DataContainer` with the table content data
  - Allocation via `setDataContainer` or constructor
- Aggregates table columns (`DoColumn`)
- To be overwritten for filling logic, if necessary:
  - `retrieve`
- Methods to be called to work with selections:
  - `getSelection`, `setSelection`, `getMultiSelection`, `setMultiSelection`



DoColumn – Table column

- Connection to a Column instance from the `DataContainer` of the superordinated `DoList`
- Connection established at the point of creation in superordinated `DoList`: bind call
- Methods correspond to those for `DoField`



DoList, DoColumn and DataContainer: Example

```
public class DoListPersons extends DoList {

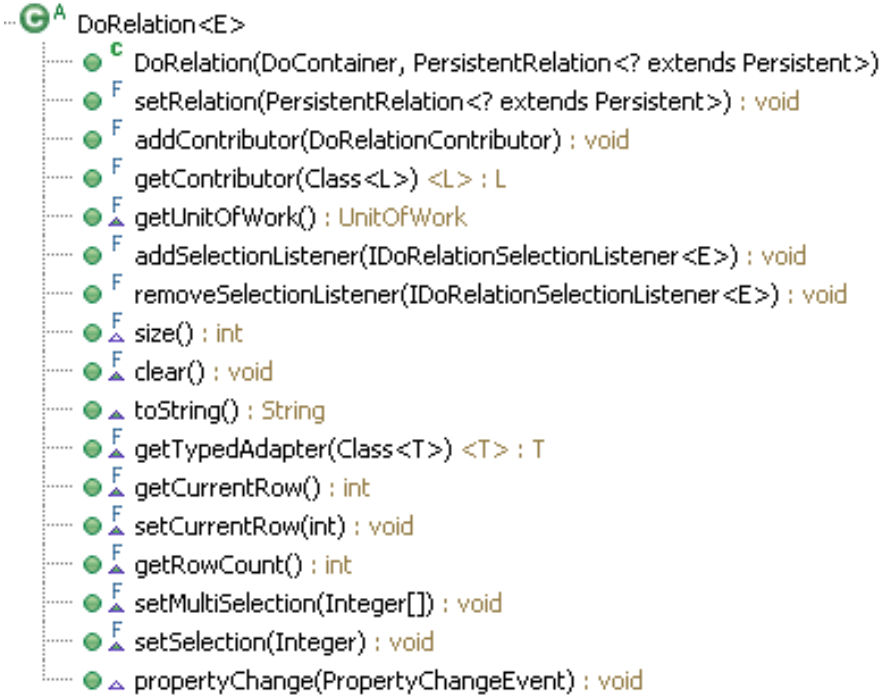
    private final DoColumn<DbDate> columnBirthday;
    private final DoColumn<String> columnName;

    public DoListPersons(DoBase parent) {
       super(parent);
       DcPersons dc = new DcPersons(getSession().getLocale());
       setDataContainer(dc);
       columnBirthday = bind(new DoColumn<DbDate>(DbDate.class),
              dc.getBirthday());
       columnName = bind(new DoColumn<String>(String.class),
              dc.getName());
    }
    public DoColumn<DbDate> getColumnBirthday() {
        return columnBirthday;
    }
    public DoColumn<String> getColumnName() {
        return columnName;
    }
}
```

DoRelation: Table, list

Linked to a `PersistentRelation` and a business object with `BoAttributes`

- Binding to `PersistentRelation` by constructor
- No DataContainer involved
- Aggregates table columns (`DoColumnAttribute`)
- Changes in the relation (new objects, deletions etc.) are handled automatically



DoRelation: Table, list

Prerequisites:

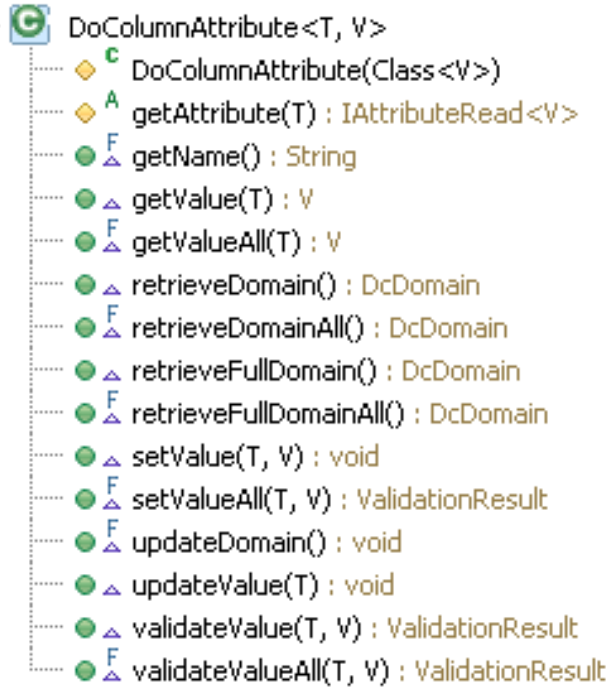
- A list based on persistent objects of a specific type should be represented
- These persistent objects are contained in a `PersistentRelation`

Building blocks:

- Business object
  - Adapts the persistent object
  - Contains a `BoAttribute` for each column
- Dialog object
  - Derived from `DoRelation`
  - Contains for each column a `DoColumnAttribute` which refers to a `BoAttribute` from the business object
  - `PersistentRelation` or `ChildPersistentRelation` needs to be passed to the constructor (or in special cases more than one)

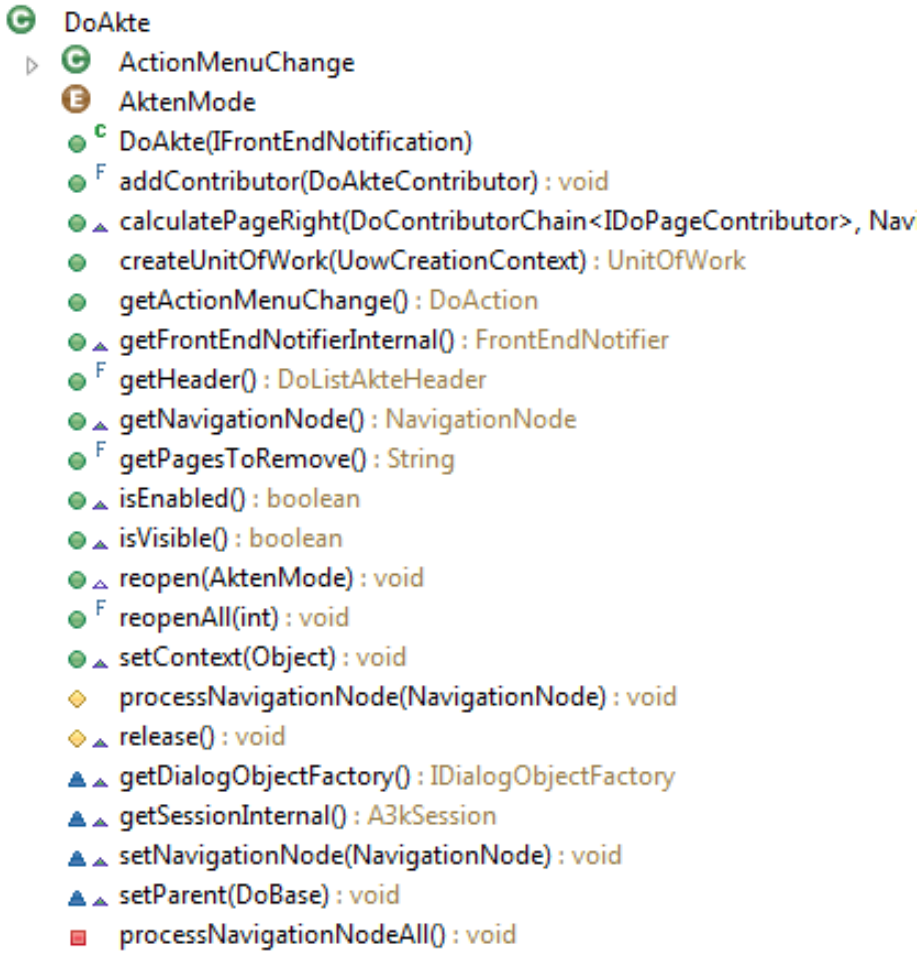
DoColumnAttribute – Table column for DoRelation

- Abstract base class
- Abstract method `getAttribute(T)` has to return a `BoAttribute`
- Connection to `DoRelation` by method `DoRelation.bind (DoColumnAttribute)`
- Methods correspond to those for `DoColumn`

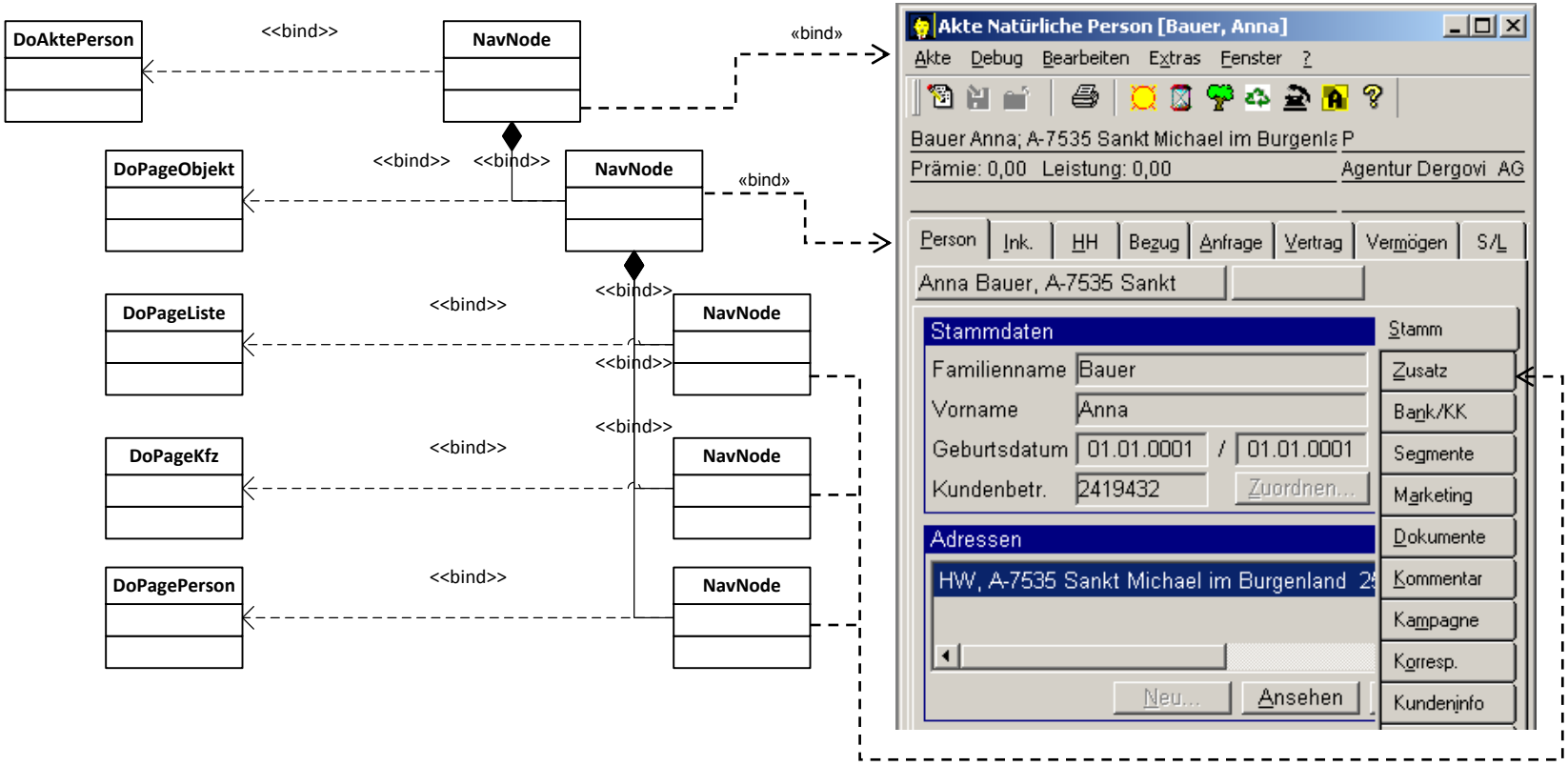


DoAkte – File window

- Defines a file window in ABS
- Method `processNavigationNode` is responsible for the creation of the navigation hierarchy, consisting of tabs and side-tabs
  (see the following slides)
- File dialog class must be mapped to a GUI class
  (see GUI layer, `RemoteClassMapper`)



Window hierarchy



Navigation structure hierarchy broken down into files, tabs, side-tabs

- Each file, tab, side-tab has a node: a `NavigationNode` object
- `NavigationNode` knows its corresponding dialog class: `DoAkte`, `DoVkPage`, `DoPage`

Window hierarchy

- Establish window hierarchy using `NavigationNodes`
  - DoAkte owns hook method `processNavigationNode`
  - The hierarchy for the subordinated NavigationNodes is established there, based on the file node
  - At least one dialog class and a logical name (for org. model) have to be defined for a NavigationNode.
    - `public NavigationNode addPage(Class<? extends DoPage> pageClass, String logicalName)`
  - Tabs/side-tabs are attached to the end using `addPage` or using `addPageAfter` or `addPageBefore` between the existing
  - Set tab/side-tab text via `setTabText`

Window hierarchy

- Window hierarchy structure via file configuration classes
  - Convention: For each new tab, a separate configuration class (`implements INavigationNodeConfigurator`) has to be created.
  - The processNavigationNode method in the file class remains manageable as a result
  - The same hierarchy can be used again in other places
  - Name convention: class name prefix `Config`

VKS pages (1)

VKS = “Verwendungsklassensicht” (usage class view):

- Group of pages on a list of entities
- Consists of:
  - VKS page: can display a header
  - List page: contains list view of the entities
  - Other pages:
    - Show details of the selected entity
    - Interested in a subset of entities in each case
    - Can be displayed or hidden depending on the internal group status

VKS pages (2)

- Usage class must consist of at least an umbrella page (`DoVkPage` successor, parent of other pages in the group) and a list page (`DoPage` successor).
- The other pages are `DoPage` successors.
- The internal group status can be changed using `configureVk(Object obj)`.
- `configureVk(Object obj)` works wherever the framework reacts to GUI updates.
- All pages in the group that can deal with the object are displayed.

VKS pages (3)

- As soon as a page is created, a notification is made as to which objects it can deal with – with certain values.
- Examples:
  - DoPageStammAuftragArag is always displayed if the selected order is type “AA” or “AP”.
    - `nodeObjekt.addPage (DoPageAuftragArag.class,"auftrag_stamm")`
      `.setTabText("Stamm").addSupportedValue("AA", "AP");`
    - `configureVk(selectedAuftrag.getAuftrart());`

Exercise 1.1.1/2: Insert search tab in the person file

- Basic structure dialog and GUI layer plug-in; add NavigationNode via DoAkteContributor
- RemoteClassMapper (is necessary to make the application runnable again, gets discussed in the GUI chapter)
  - In the person file, a new tab called “Search” should be inserted at the last position. It should be accessible using the keyboard shortcut “S”.

Exercise 1.1.3/4: Insert criteria side-tab; Exercise 1.1.5/6: Insert result side-tab

- Repeat: Insert NavigationNode
- WPage derivation (still empty); RemoteClassMapper entry
  - In the person file, in the search tab, a new side-tab called “Criteria” should be inserted.
  - In the “Search” tab, a further side-tab called “Result” should be inserted at the very right.

Context information

- Dialog objects can require different pieces of context information, e.g.:
  - The primary object of the current file
    (e.g. the person opened in the person file)
  - The object selected in a superordinated VKS list
    (e.g. the contract selected in the contracts list)
- Context information is required for the complete initialisation of a page:
  - Business objects can be fetched
  - Subordinated dialog objects can be initialised

- Be aware:
  - Inside of a VKS group, the context information changes if the list selection changes
  - In all other cases, the context information gets passed only once

Context information (2)

A3k offers three concepts to handle context information:

- Dependency injection (introduced with Core 14.5)
- Context objects (frequently used in existing code)
- PageParameters (legacy technique, used in isolated cases)

Context objects (1)

- The context object is placed into the page dialog object (`DoPage`) by the framework using the `setContext` method call
- The `DoPage` is responsible to pass this context information to subordinated objects, if required
- The context object usually describes or refers to a persistent object

- Context objects can be (depending on the GUI implementation):
  - A subtype of `Context` (used in most cases, for historical reasons), e. g. `ContextGschfall`:
    - Contains a primary key (`DBKey`) and a timestamp
    - Method `getGeschfall` fetches the corresponding persistent object
  - An object of any type, e.g. a persistent object

Context objects (2)

Example: Context subtype

```
@Override
public void setContext(Object context) {
    super.setContext(context);
    ContextGschfall contextGschfall = (ContextGschfall) context;
    Gschfall gschfall = contextGschfall.getGschfall(getUnitOfWork());
    if (gschfall instanceof Person) {
        Person person = (Person) gschfall;
        // …
    } else if (gschfall instanceof Vertrag) {
        Vertrag vertrag = (Vertrag) gschfall;
        // …
    }
}
```

Context objects (3)

Example: Object

```
@Override
public void setContext(Object context) {
    super.setContext(context);

    if (context instanceof Person) {
        Person person = (Person) context;
        // …
    } else if (context instanceof Vertrag) {
        Vertrag vertrag = (Vertrag) context;
        // …
    }
}
```

Dependency injection

The framework injects context information into dialog objects annotated with `@Inject`

- General prerequisites:
  - The current file or VKS list is configured for dependency injection: `DoInjectionConfigurator` needs to be attached (Prepared files in Core 15.0: Person, claim)
  - Dependency injection is switched on for the current `NavigationNode`: `navigationNode.setInjectionEnabled(true);`
- Annotations in dialog classes:
  - `@Inject` at the constructor (mandatory)
  - `@Inject` at field variables or methods which take the context information
  - `@PostConstruct` at a method to be executed after the complete injection
- Create subordinated objects which use dependency injection with `createContainer(DoClassName.class)`

Dependency injection: Example (1)

- Fields and methods marked with `@Inject` get provided with context information
- The constructor always needs to be annotated with `@Inject`

```
public class DoPageSchadenExample extends DoPage {

    private DoGroupSchadenExample doGroupSchadenExample;

    @Inject
    private Schaden schaden; // This field will take the claim object

    private BoSchadenExample boSchadenExample;

    @Inject
    public DoPageSchadenExample(DoPage parent, UnitOfWork unitOfWork,
            IFrontEndNotification notification) {
        super(parent, unitOfWork, notification);
    }

    …
```

Dependency injection: Example (2)

- A method marked with `@PostConstruct` gets executed after the injection process and can take initialisation code

```
…

@PostConstruct
private void initialise() { // Or a random method name

    // Use the injected persistent to fetch the business logic:
    boSchadenExample
            = schaden.getCheckedAdapter(BoSchadenExample.class);

    // Initialise a subordinated dialog object:
    doGroupSchadenExample
            = createContainer(DoGroupSchadenExample.class);
    …
}
```

View/change mode

- On file window level
  - Is automatically put into the `DoAkte` dialog instance by the framework
  - Query using `isAkteEnabled()`
- On page level
  - Is automatically put into the `DoPage` dialog instances by the framework
- Subordinated dialog objects
  - Can query the change mode using `isPageEnabled()`
  - Automatically inherit the change mode of the superordinated object in the `isEnabled()` method

Validation

- Validations are initiated by the framework via the dialog objects:
  - `DoField` and `DoColumn`: field validation
    - Framework first of all calls `validateValue`
    - Only after successful validation, this is followed by `setValue`
- `DoPage`: Page validation
  - Framework first of all calls `validate`
  - Only after successful validation, this is followed by `deactivate, save`, etc.
- Dialog objects do not implement the validation directly, but delegate to the business logic:
  - To `validateValue` methods of the business attributes
  - To other `validate…` methods of the business objects

Updating the GUI

- Fine-grained GUI updates can be performed:
  - Dialog object base class `DoContainer` offers methods to update the object, including all subordinated objects:
    - `retrieveValues()`
    - `retrieveVisible()`
    - `retrieveEnabled()`
    - `retrieveDomains()`
  - Page and file permissions can be updated via `IFileManager`:
    - `refreshPageRights()`
    - `refreshFileRights()`
    - `refreshPageListRights()`
- Hook method for page activation, overwrite if necessary: `activate` in `DoPage`

Legacy class – FrontEndNotifier

- Only for PB GUI
  - Serves to store information/references for the GUI
  - The following references can be set:
    - Update the visible properties of all controls
    - Update the enabled properties of all controls
    - Update the data for all controls
    - Update the page rights (current file)
  - Works asynchronously, i. e. not always immediately
  - Is evaluated based on certain framework calls:
    - `DoAction.execute()`, `DoList.setSelection()`, `DoColumn.setValue()`, `DoField.setValue()` etc.
    - Not after user-defined methods

Page rights (1)

- Right = min(application right, org. model right)
- Org. model right is always requested by the framework.
- Parent page is responsible for its children.
- The right for a page is requested by its parent first. Parent passes the call on to contributors (takes A3k distance into account).
- If it is not found, the parent passes the call on to his parent, up to file level. The file delivers `PageAccess.ALL` as standard.

Page rights (2)

- Right is calculated in the hook method `calculatePageRight`.
  - `public PageAccess calculatePageRight(DoContributorChain<IDoPageContributor> chain, NavigationNode node)`
- `node` is the page position. The logical page name is obtained using `getLogicalName`.
- Usage class right is requested by the umbrella page for the usage class group (DoVkPage)
- The right is calculated in the hook method `calculateVkRight`.
  - `public PageAccess calculateVkRight(DoContributorChain<IDoVkPageContributor> chain, NavigationNode node, Object selection)`
- Parameter `selection` is the object with which the usage class group was configured (`configureVk(selection)`).

Exercise 1.1.7: Prepare group with input fields, drop-down list box, buttons

DoGroup, DoField, DoCommand

- On the “Criteria” side-tab the following controls should be positioned:
  - two input fields, first name and surname (deactivated for typing),
  - a drop-down list box for academic title and
  - two buttons to search and to reset the search criteria.

Exercise 1.1.8: Prepare results table

Implement table (DoList) in the dialog layer

- In the side-tab “result”, a table should be displayed listing the result of the search from the “criteria” side-tab

Quiz 1

Quiz 1 – Dialog Layer Basic Concepts (1)

- **What are the responsibilities of the Dialog Layer?**
  a) It acts like a business logic façade
  b) It's responsible for drawing the GUI
  c) It's abstracting GUI elements
  d) It's connecting ABS to the database
- **Which of the following classes are A3k Dialog Layer classes?**
  a) DoString
  b) DoField
  c) DoRelation
  d) DoCommandAttribute
- **What other A3k classes can be accessed via DoBase?**
  a) UnitOfWork
  b) ValidationManager
  c) BoBase
  d) FrontEndNotifier
- **Which GUI elements can be abstracted using DoField**?
  a) Dropdown Listbox
  b) Button
  c) Label
  d) Checkbox

Solution: a, c – b, c – a, b, d – a, c, d

Quiz 1 – Dialog Layer Basic Concepts (2)

- **What is the difference between DoList and DoRelation?**
  a) There is no difference between those two
  b) DoList requires a DataContainer, DoRelation doesn't
  c) DoRelation allows multi-selection, DoList doesn't
  d) DoRelation replaced DoList in core 15.0
- **What applies to a DoColumnAttribute?**
  a) It is used to define columns in a DoList
  b) It is used to define columns in a DoRelation
  c) It is an abstract base class and needs to be subclassed
  d) Such a class doesn't exist in the A3k framework at all
- **What can a NavigationNode represent?**
  a) A tab
  b) A button
  c) A side-tab
  d) A popup window
- **What belongs to a VKS page?**
  a) A list page
  b) A popup
  c) Other pages that show details of the selected entity
  d) The ABS main screen

Solution: b – b, c – a, c – a, c

Quiz 1 – Dialog Layer Basic Concepts (3)

- **What types of validation are available in ABS?**
  a) Field validation
  b) Page validation
  c) File validation
  d) ABS-wide validation
- **DoContainer offers methods to update dialog objects, including:**
  a) retrieveValues
  b) retrieveValidations
  c) retrieveData
  d) retrieveEnabled

Allianz

GUI code and plug-in structure

A3k GUI framework code:

- Plug-ins `at.allianz.framework.core.gui` and `at.allianz.framework.core.gui.swt`

GUI code of the core application:

- Plug-in `at.allianz.gfb.core.gui`

Customer-specific code:

- Target structure: GUI code in the same plug-ins as the dialog logic code, functionally separated plug-in structure
- Or one monolithic GUI plug-in, as result of the translation of the existing PowerBuilder code
  (e.g. AEV: `at.allianz.gfb.aev.gui`)

Allianz

SWT basics

- SWT: API for platform-independent GUI programming
- Originally developed for Eclipse, as an alternative to the standard API Swing
- Its user interface controls are called widgets
- Various widget classes available:
  - Simple widgets: E.g. `Label`, `Text`, `Button`
  - Containers: E.g. `Composite`, `Group`, `Shell`
- Container widgets can contain subcomponents
- Each constructor has two parameters, e.g.
  `public Label(Composite parent, int style)`
  - `parent` defines the hierarchical connection from child to parent
  - `style` defines the look of the control

SWT layout

- Each SWT widget is responsible for a rectangular area which has a specific position and size
- Simplest approach: Absolute positioning using pixel data
  Disadvantage: Does not adopt to font settings and window size
- Better and mandatory for ABS: Use of a layout class (‘layout manager’)
- Layout can be associated to a container and is then responsible to lay out all subordinated elements
- Most important layout types for ABS:
  - `FillLayout`: Elements get arranged in one row, either horizontally or vertically; each gets the same size
  - `GridLayout`: Elements get arranged in a grid with columns and rows; elements can have `LayoutData` attribute with complex parameters
- Also used sometimes: `FormLayout`, `RowLayout`

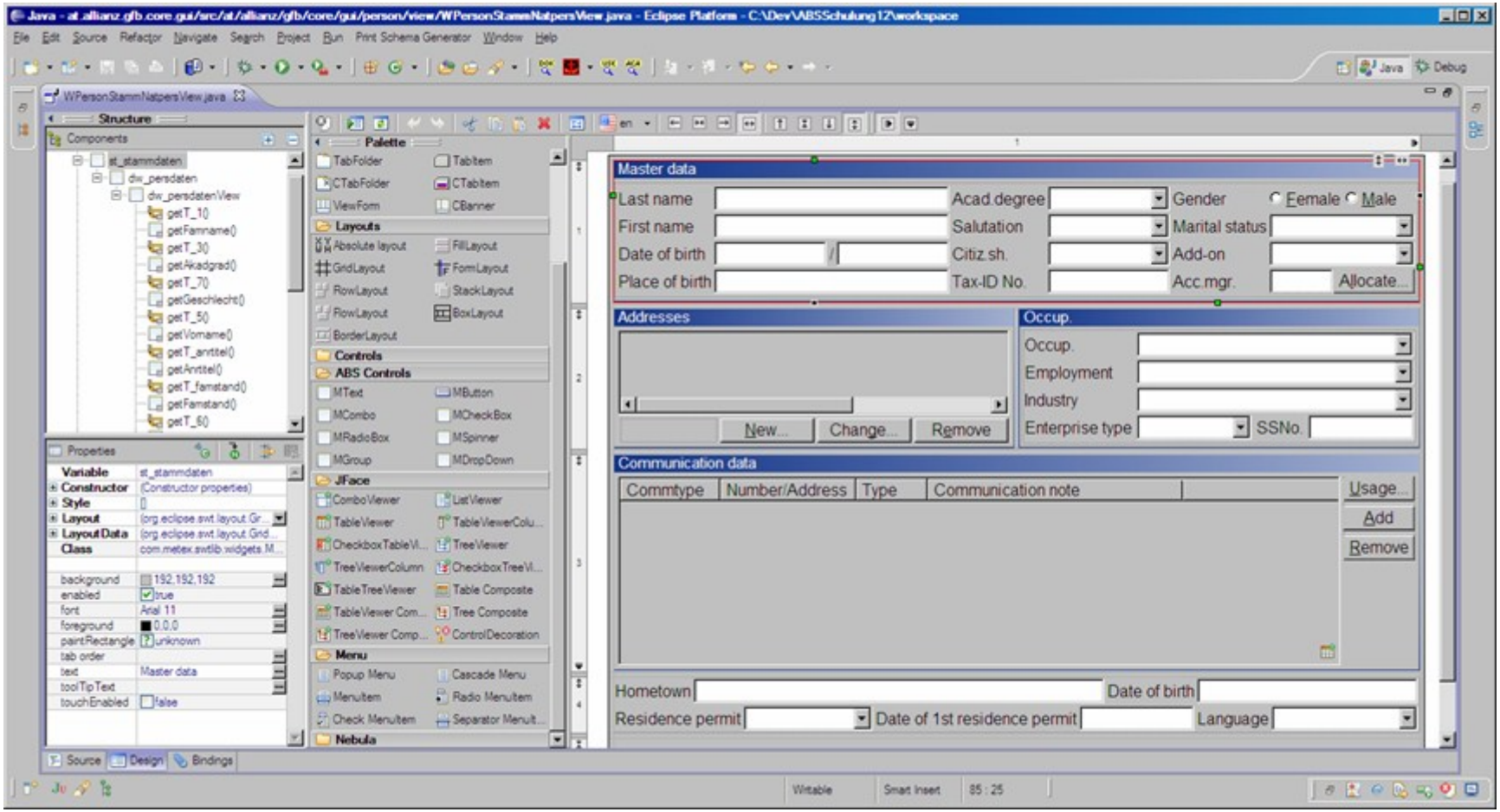
SWT, JFace and ABS controls

Controls used to create the user interface

- Standard SWT widgets
  - `Composite, Label`
- JFace controls:
  - `TableViewer, GridTableViewer, GridTreeViewer`
- ABS controls:
  Widgets with ABS-specific characteristics (appearance, size, behaviour), following the ABS style guide
  - `MButton, MCheckBox, MDropDown, MGroup, MRadioBox, MSpinner, MText`

SWT Designer

Eclipse IDE plug-in: SWT Designer (part of Eclipse WindowBuilder)



Component structure

The user interface consists of nested components

- **Page** component:
  - A screen page which fills one tab of a file window or a pop-up window
  - It contains one or more subordinated components
- Subordinated component: can either be
  - A **group** component (form-like composition of widgets)
  - A **list** component (table or grid)
  - A **tree** component

Component structure and layout

View class

- Contains the **visual representation** of a user interface component, must not contain event handling or other logic
- Class name convention: `Prefix Page, Group, List` or `Tree`; suffix `View`
- Is derived from the SWT class `Composite`, Page component gets derived from `WPageView` (a subclass of `Composite`)
- Each component needs to be put in a separate view class
- Public get methods for all widgets

- Page view contains a **placeholder** for each subordinated component:
  - Type `MGroup` or `Composite`
  - Layout must be set to `FillLayout`
  - Has to contain only the subordinated view (WYSIWYG in SWT Designer) or can also be empty (then the view gets instantiated by the framework)

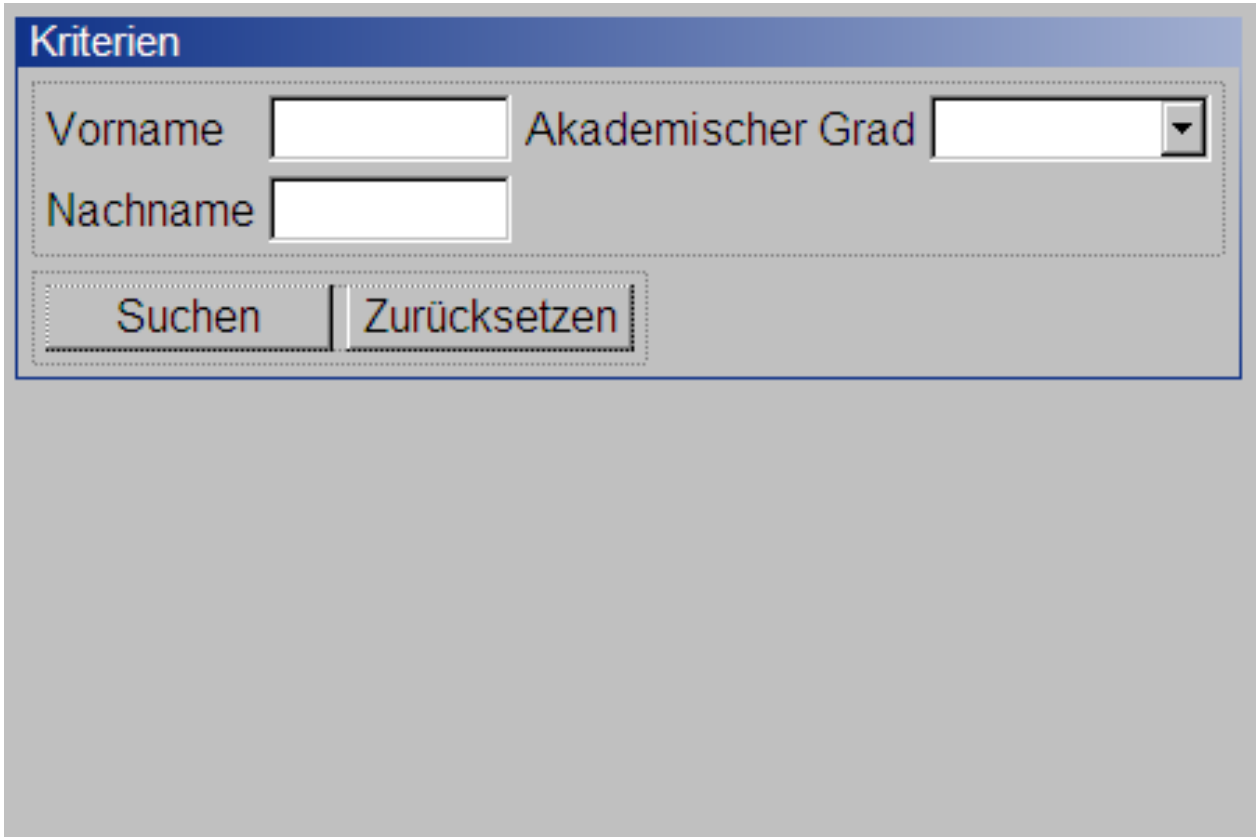
SWT details

- Test automation:
  - A **technical name** is required for all controls which get connected with the Dialog Layer, e.g.:
    - ```
      MText text = new MText(composite, SWT.NONE);
      text.setData(Constant.NAME, "text");
      ```
- Layout
  - Different font and window sizes need to be supported
    → The layout needs to be dynamic, never use absolute (pixel) layout
  - `GridLayout` and `FillLayout` are recommended in most cases
  - Avoid ‘magic numbers’ for pixel distances and over-constraining
  - Labels have to use the font returned by `FontFactory`:
    - `label.setFont(FontFactory.getLabelFont());`
  - Radio buttons always have to be placed in an `MRadioBox`
      ```

Exercise 1.2.1: Create view classes of the criteria page

Create view classes with SWT

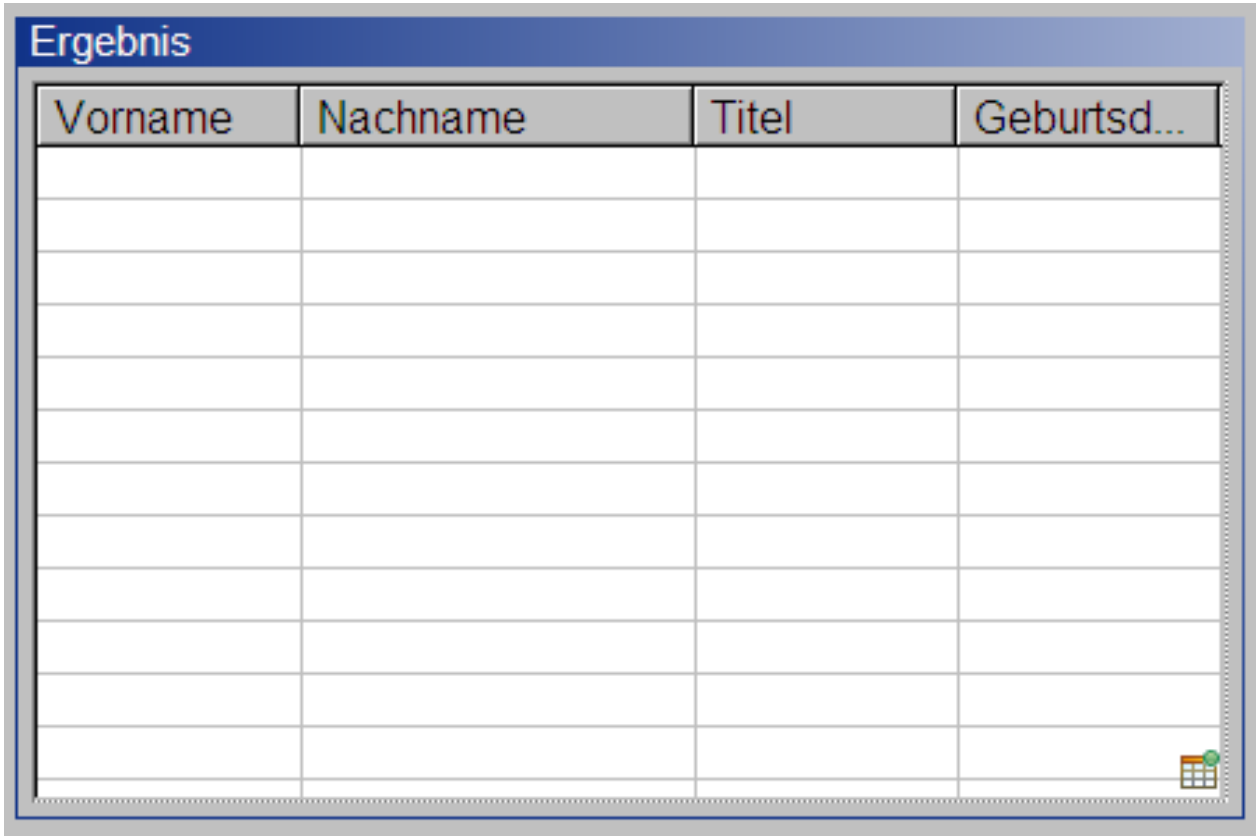
- The criteria page should look like this:



Exercise 1.2.2: Create view classes of the results page

Create view classes with SWT

- The results page should look like this:



Java GUI versions

How to connect the GUI with the dialog layer?

- Current framework version, Java GUI 2: Data Binding
  - Based on Eclipse Data Binding (formerly known as JFace Data Binding)
  - Available since core 10.5

- Legacy, Java GUI 1: PowerBuilder emulation (Metex framework, PB GUI)
  - Resulted from the migration of the PowerBuilder GUI code (by the company Metex)
  - Reproduces PowerBuilder concepts

Allianz

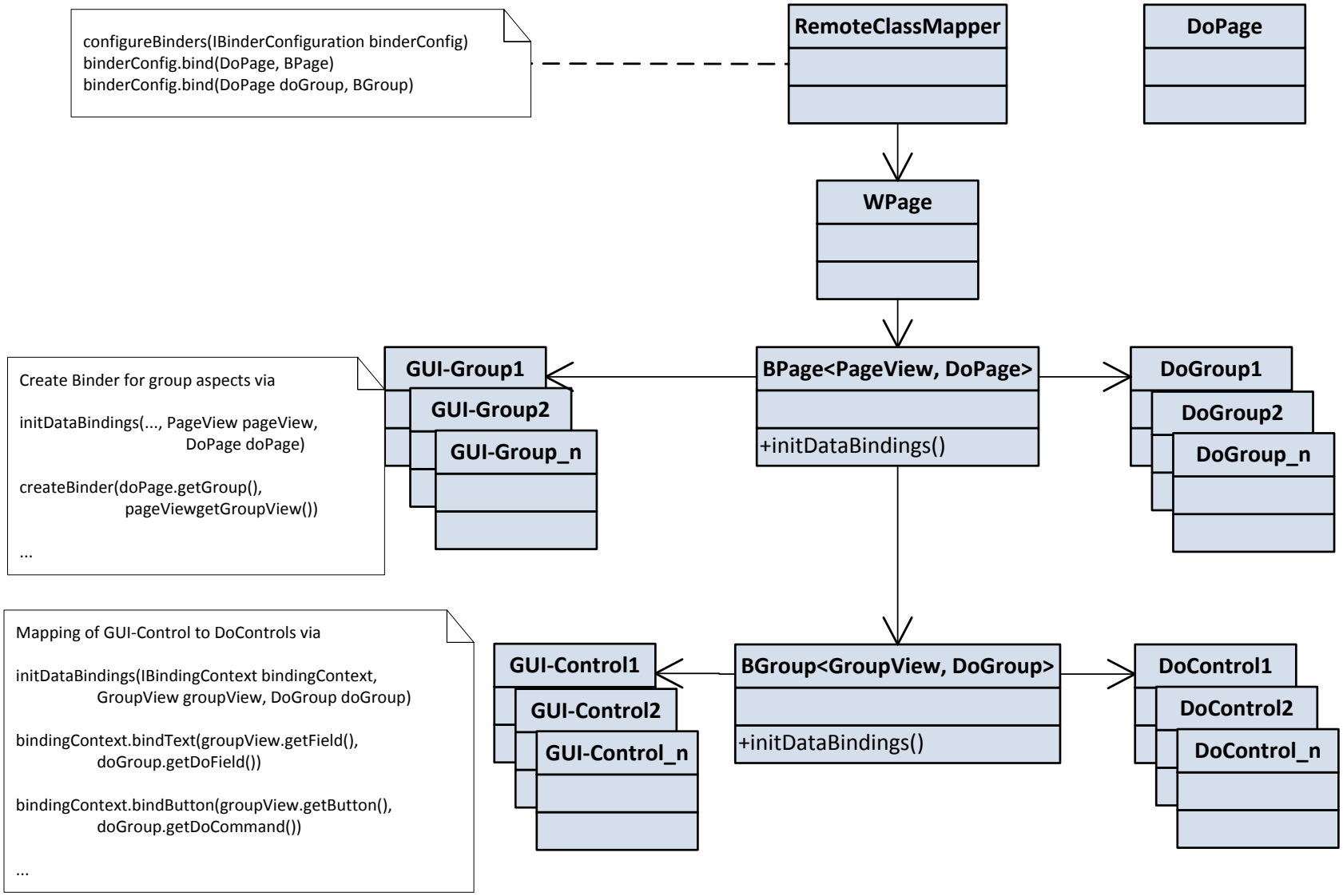
Data Binding GUI: Overview

Parts of a GUI page to be implemented

- **Component idea**: GUI consists of components,
  page, group, list or tree
- Necessary for each single component:
  - **View** class (`Composite` derivation): graphical display
  - **Dialog** class (`Do`…derivation): façade of the business logic
  - **Binder** class (`Binder` derivation): connection between the two

- Once per page:
  WPage class (`WPage` derivation): binding configuration

Data Binding GUI: How-to



Data Binding GUI: Steps to be implemented (1)

- Binder class
  - Derived from the class `Binder<T extends Composite, U extends DoContainer>`
  - Responsible for bindings of a specific component (page, group or list)
  - Registers event handling
  - Name convention: prefix `BPage, BGroup or BList`
  - Structure:
    - `initDataBindings`: Widget connections
      For each widget contained, one type-specific `BindingContext` bind call
    - `activateHandlers`: Event handling
      To activate an `A3kHandler for a DoCommand`

Data Binding GUI: Steps to be implemented (2)

- BindingContext: Binding functionality
  - To be used in `Binder.initDataBindings`
    - Offers different bind methods for various dialog object and SWT widget types, e.g.
      - `context.bindText(getView().getNameText(), getDialogObject().getNameField())`
    - Returns specific `BindingContext` types in each case, including:
      - `DoField: → IControlValueBindingContext`
      - `DoList: → ITableViewerBindingContext`
      - `DoColumn: → ITableViewerColumnBindingContext`
    - This makes it possible to set properties, e.g. `setEditMask(…)`

Data Binding GUI: Steps to be implemented (3)

- A3kHandler: Event handling base class
  - To be used in `Binder.activateHandlers`
    - Subclass should be an inner class of the binder
    - Must implement an execute method for the logic
    - Performs GUI activities or delegates to the dialog layer
    - Should not call the business logic directly
- IFileManager: Helper for GUI activities
  - To be used within an A3kHandler
    - Offers various GUI activities relating to the file window
      - Jump to other pages in the file window: `goToTab, goToSideTab`
      - Open a pop-up dialog: `openPopup`
      - Save or close the file window: `save, close`
    - Is offered by the binder base class via `getFileManager`

Data Binding GUI: Steps to be implemented (4)

- WPage class
  - Is derived from the A2k GUI class WPage
  - Is responsible for the configuration of the bindings for all components on the page
  - Name convention: prefix `W`
  - Structure:
    - `configureBinders`: Contains a call for all components of a page
      - `binderConfiguration.bind(dialogObject, BinderClass.class);`
    - Helper method `getDialogObject` should return the dialog object in typed fashion

GUI: RemoteClassMapper

- RemoteClassMapper: Connection on page level
  - Assigns a DoPage subclass to a WPage subclass
  - Wherever a NavigationNode refers to a dialog page class, the assigned GUI page gets displayed
  - Mapper class needs to be registered at the extension point `at.allianz.a3k.gui.remoteClassMap`, specifically for a file window

```
public class RemoteClassMapperAuftrag implements IRemoteClassMapper {

    @Override
    public void map(RemoteClassMapping mapping) {

        mapping.map(DoPageAuftragListeSchaden.class,
                WAuftragListeSchaden.class);

    }

}
```

Exercise 1.2.3: Define the bindings for the criteria page

Use Data Binding on the page with DoGroup

- The "Criteria" page should be implemented with Data Binding.

Exercise 1.2.4: Define the bindings for the results page

Use Data Binding with DoList

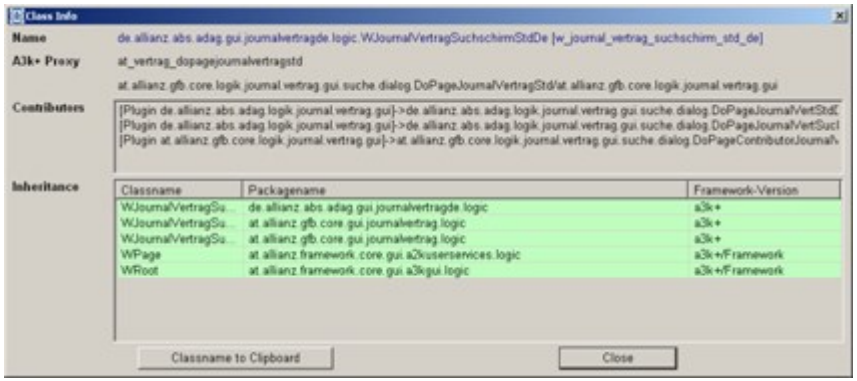
- The "Results" page is also to be implemented with Data Binding.

Debug functions (1)

- Precondition: application start parameter /DEBUG
  - Helpful keyboard shortcuts for Data Binding:
    - Ctrl + Alt + B
      shows all `Binders` of the page
    - Ctrl + Alt + C
      shows all `DoCommands` (and `DoActions`) which are bound to handlers
    - Ctrl + Alt + L
      shows for all `DoLists` the `DataContainer` content

Debug functions (2)

- Right mouse click
  provides information on the GUI page class name and the dialog class name



- Ctrl + Shift
  provides information on the GUI element under the mouse cursor
  and its widget hierarchy

```
controlclass                          visible  controlname          tabindex content
------------                          -------  -----------          -------- -------
Shell                                       1  null                        0
 WJournalSuchschirmPopupDeView              1  null                        0
  ScrolledComposite                         1  null                        0
   Composite                                1  null                        0
    MTabFolder                              1  tab_0                   60000
     Composite                              1  null                        0
      WTabpagelistView                      1  null                        0
       MVerticalTabFolder                   1  tab_1                   60000
        Composite                           1  null                        0
         WJournalVertragSuchschirmStdDeView 1  null                        0
```

Quiz 2

Quiz 2 – GUI Layer Basic Concepts (1)

What types of controls can be used to create a GUI in ABS?

a. Standard SWT widgets
b. AWT widgets
c. JFace controls
d. ABS controls

Which tool can be used from within Eclipse to design GUIs?

a. GUI Wizard
b. SWT Composer
c. Window Designer
d. SWT Designer

The GUI consists of nested components. Those can be:

a. Group components
b. Page components
c. Pop-up components
d. Desktop components

SWT supports so-called layouts to align controls on the screen. Common layouts are:

a. GridLayout
b. SpanLayout
c. FillLayout
d. LabelLayout

Solution: a, c, d – d – a, b – a, c

Quiz 2 – GUI Layer Basic Concepts (2)

What types of classes are part of the GUI layer?

a. Binder class
b. View class
c. Dialog class
d. Persistent class

What is the purpose of a binder class?

a. It is responsible for bindings of a specific component
b. It draws the GUI
c. It provides access to the business layer
d. It registers event handling for controls

What applies to a WPage class?

a. It is responsible for the binding configuration
b. It defines the SWT controls to be used
c. It serves as the controller of a page
d. It provides the business API for a page

What is the prerequisite to use debug functions in ABS?

a. The machine that runs ABS must have at least 8 GB of RAM
b. The application start parameter /DEBUG must be provided
c. ABS code must be compiled with debug information
d. At least ABS core 14.5 must be used

Solution: a, b – a, d – a, c - b

General requirements

- Flexible adjustment of the GUI to suit country-specific standards
  - Language
  - Display features
    (e.g. date, time, currency formats, etc.)
- Simple modification and extension options
  - Relocation to text files

Allianz

Support provided by SWT Designer

- Wizard support for new locales
  (property files and access classes)
- Preview for various languages
  (allows to switch language quickly)
- Export of existing control texts to property files
  (labelling, tool tips, …)
- Straightforward changes to properties on the GUI per locale

Technical details/special features

- Java: Distinction made using Locale object
- SWT Designer
  - Creation of access class (`Messages`)
  - Creation of a property file for each locale
  - Change to text allocation in the source code (allocation via `Messages` class)
- Conventions
  - Only one `Messages` class and one property file per locale and package (applies to `core.gui` and functional packages)
  - Use the “Classic Eclipse messages class” option
  - When Core messages should get overwritten by customers, fragment bundles have to be used (→ section on extensions)

ABS-specific Messages classes

The generated Messages class needs to be adjusted, as the `Locale` setting of the current ABS session has to be used

- GUI and dialog layer (Rich Client):
  - Messages class uses the A3k class `ResourceBundleProviderUI` which uses the static `Locale` setting
  - See examples in plug-in `at.allianz.gfb.core.gui`
- Business logic:
  - Messages class uses the A3k class `ResourceBundleProvider`
  - Method `getText` has a second parameter for the `Locale`
  - Caller passes the current session Locale `getSession().getLocale()`
  - See examples in plug-in `at.allianz.a3k`

Exercise 1.3.1: Relocate user interface texts

Relocate text to properties file

- All strings that are visible on the user interface are to be relocated to properties files.

Allianz

2

1 Basic concepts (core development)
- 1.1 Dialog layer
- 1.2 GUI layer
  - 1.2.1 Structure and tools
  - 1.2.2 Data Binding
- 1.3 Internationalisation

2 Extension procedure (customer-specific adjustments)
- 2.1 Dialog layer
- 2.2 GUI layer
- 2.3 Internationalisation

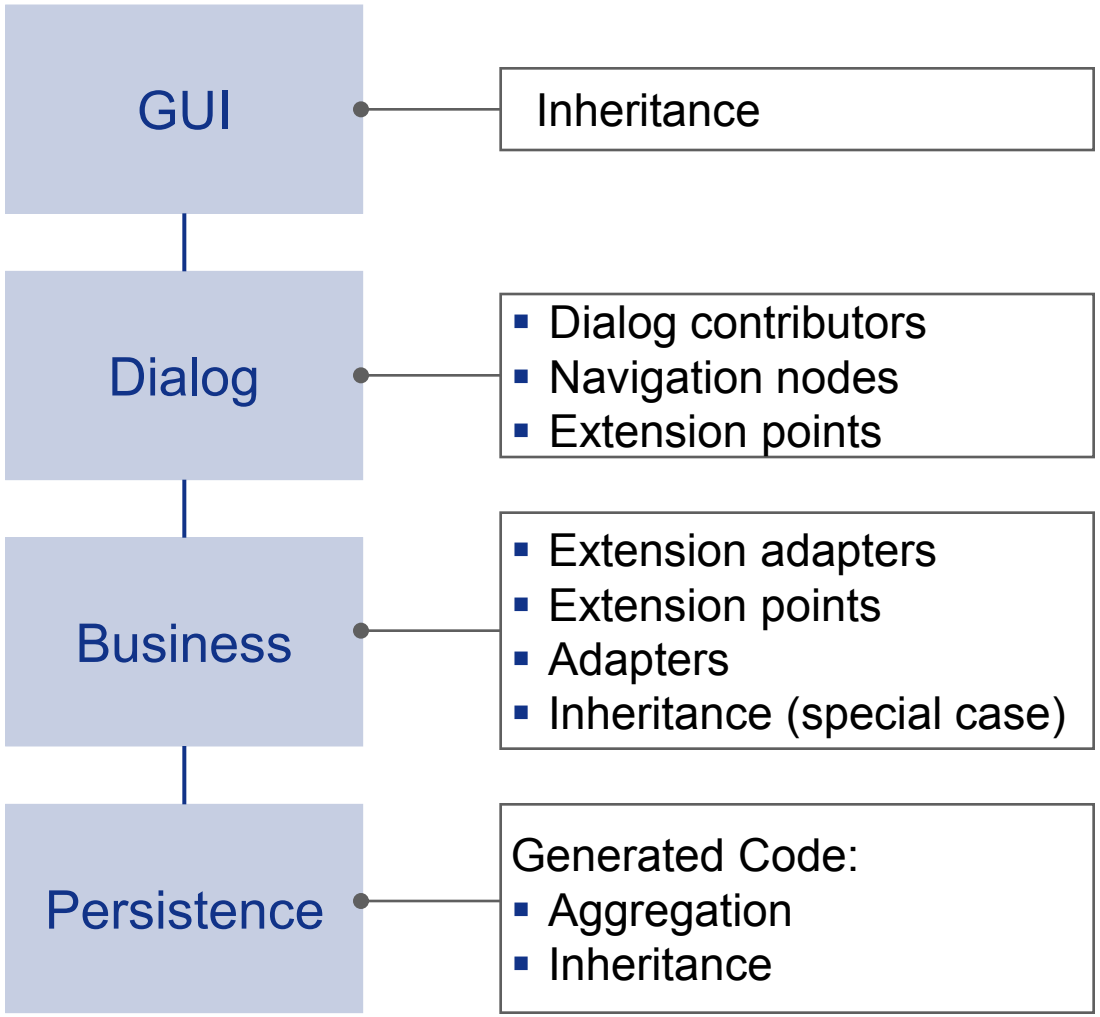
3 Cross-functional topics
- 3.1 Configuration
- 3.2 Logging

ABS extension concepts – configuration

- Objective: change core behaviour to implement customer requirements
  - Two options: using parameters or programme code
    - Parameterisation
      - Alternative scenarios coded in the core that can be selected using a switch
      - Possible implementation via defining data tables, e.g. `TFIRMENDATEN` or configuration switch

ABS extension concepts – programming

Core code must not be changed by the customer!



Allianz

2

Dialog layer extension links

- Concept
  - Powerful generic extension links
  - Almost the entire communication between the dialog layer and business logic can be intercepted and overwritten by the customer
- Examples
  - Visibility, enabled/disabled, authorisation for controls, columns and pages
  - Data acquisition (incl. domains) for displaying controls
  - Provision of additional columns or hiding columns
  - Activity (e.g. when a button is clicked), validation

Extension points

- Extension Point: `at.allianz.a3k.gui.dialogContributor`
  - Serves to register dialog contributors
- Extension Point: `at.allianz.framework.core.gui.fileDef`
  - Serves to register files
- Extension Point: `at.allianz.a3k.gui.remoteClassMap`
  - Serves to register class mappings (NavigationNode/dialog object → Page)

Allianz

Dialog contributors (1)

- Instead of inheritance paradigm → aggregation of dialog entities (objects which contribute functionality to the original objects)
- A contributor object can be attached to a core dialog object in order to change its behaviour
- The contributors are linked via extension points and the API
- In the rare case when more than one contributor is attached to the same dialog object, the call order is based on plugin distance from framework plugins

Dialog contributors (2)

- Extensions for dialog objects:
  - Interception/overwriting of method calls
    - E.g. isEnabled(), isVisible(), retrieve(), ...
    - Execution order based on plug-in distance to the framework: Greater distance means higher priority
  - Structural extensions of the dialog object
    - Addition of new columns, fields, actions etc.

Dialog contributors (3)

- Each framework dialog class has a specific contributor class
  - E.g. `DoField` → `DoFieldContributor`
  - `DoPage` → `DoPageContributor`
- Registration
  - `DoPage, DoAkte` – via extension point
  - Controls – via hook method `DoContributorBase.mapContributors`
  - `DoGroup, DoList, DoTree` – either of those methods (but not both at once)

Dialog contributors (4)

- Example

```
public class DoPageContrAB extends DoPageContributor<DoPageAB> {

    private class DoColumnContrXY extends DoColumnContributor<String> {

        public final String getValue(Row row,
                DoContributorChain<IDoColumnContributor<String>> chain) {
            ...
        }
    }
    public void mapContributors() {
        getDialogObject().getList().getColXY().addContributor(
                new DoColumnContrXY());
    }
}
```

Window hierarchy (1)

- Extension
  - Overwrite `DoAkteContributor.processNavigationNode`
    - Attributes can be changed later on (navigate to the desired NavigationNode using `getNode(<dialog class>, <logical name>)` and call the respective setter method)
    - Contributors can add new NavigationNodes using the corresponding add method `(add, addAfter, addBefore)`
    - `NavigationNodes` can be removed using the `removeNode` method.

Window hierarchy (2)

- Example

```
public class DoAkteContrPersonHelloWorld extends
    DoAkteContributor<DoAktePerson> {
     protected void processNavigationNode(NavigationNode node) {
         // Fetch the predecessing node
             NavigationNode nodeInkasso =
               node.getNode(DoVkPageInkassoPerson.class,
               PageConstPerson.PERSON_VK_INKASSO);
         // Position the node
             NavigationNode nodeHello = node.addPageBefore(
               DoPageHelloWorld.class, "HELLO_WORLD", nodeInkasso);
         nodeHello.setTabText("&Hello");
     }
}
```

Exercise 2.1.1: Change in visibility via contribution

DoListContributor, DoColumnContributor

- The "title" column is no longer to be displayed in the table of the results page.

Exercise 2.1.2: Extending a table

New DoColumn using DoListContributor

- On the "Search/Result" page, the results table should be extended to include the column "age" (“Alter”) in order to meet the customer's specific needs.

Exercise 2.1.3: Adding additional dialog logic

Additional dialog logic in a contributor class

- The dialog layer of the search criteria group should get a method which calls the extended search method of BoNatpersSearchDe. (Later, this method will get called from the GUI.)

Quiz 3

Quiz 3 – Dialog Layer Extension Concepts

What extension mechanisms can be used at the dialog layer?

a. Contribution
b. Inheritance
c. Extension points
d. Extension adapters

Why would a customer want to extend the dialog layer?

a. He wants to introduce additional columns in a table
b. He wants to change the business logic
c. He wants to alter field validation
d. He wants to alter the visibility of a control

What can a customer achieve using dialog contribution?

a. He can intercept the isEnabled method
b. He can hide core columns in a table
c. He can alter the business logic
d. He can introduce new persistent objects

The A3k framework provides contributor classes for each type of dialog class, for example:

a. DoFieldContributor
b. DoLabelContributor
c. DoButtonContributor
d. DoPageContributor

Solution: a, c – a, c, d – a, b – a, d

Allianz

2

Allianz

2

2 Extension procedure (customer-specific adjustments)

2.1 Dialog layer

2.2 GUI layer

2.2.1 Visual aspects (views)

2.2.2 Data Binding

2.2.3 PowerBuilder emulation basics

2.2.4 PowerBuilder emulation extension

2.3 Internationalisation

GUI extension links

- Concept
  - Dependent on the GUI technology
  - SWT → inheritance from components/pages

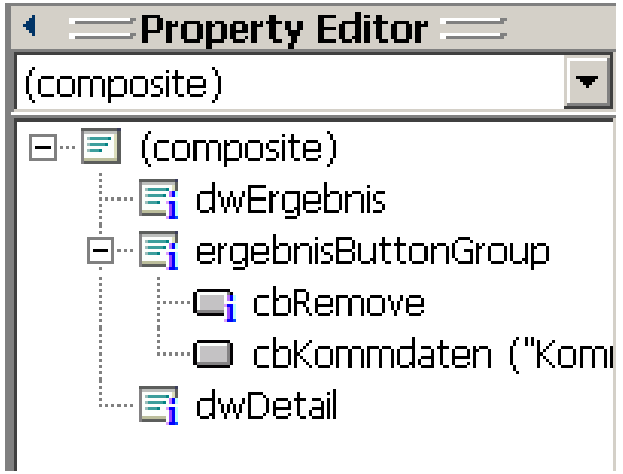
- Examples
  - Adding controls by way of page component inheritance
  - Adding columns by way of table component inheritance

Extending a page (1)

- Objectives
  - Modify controls (change colours, etc.)
  - Add controls
  - Remove controls
  - Change event handling
- Strategy
  - Create page successor
  - Replace original page with the successor

Extending a page (2)

- Modify controls – visual features
  - Control has to be accessible for the successor (public getter method)
  - Possible using SWT designer
  - Controls inherited from the predecessor are marked in the Property Editor



NB: This is not a wise method for altering text!
Instead, alter text using .properties file → Internationalisation

Extending a page (3)

- Add controls
  - Random positioning not possible due to the layout manager in use
  - At the end, attachment possible for `FillLayout`, `RowLayout` and `GridLayout`
  - Placeholders (Composites) have to be used explicitly in the core page in the places to be extended
  - Convention: each button and each component should be kept in a Composite or in an `MGroup`

Extending a page (4)

- Remove controls
  - Make them invisible via the dialog layer
    (to be used where possible)
  - On the GUI side: At `GridLayout`, `RowLayout` by setting the layout data attribute `exclude` to `true`

Extending a page (5)

- Replace original page with the successor:
- Map NavigationNode/dialog object to the new GUI page class (RemoteClassMapper)

```
public class RemoteClassMapperNewTabExtensionPerson
        implements IRemoteClassMapper {

    @Override
    public void map(RemoteClassMapping mapping) {
        mapping.map(DoPageResult.class, WResultExt.class);
    }

    …
}
```

2

2 Extension procedure (customer-specific adjustments)

2.1 Dialog layer

2.2 GUI layer

2.2.1 Visual aspects (views)

2.2.2 Data Binding

2.2.3 PowerBuilder emulation basics

2.2.4 PowerBuilder emulation extension

2.3 Internationalisation

Data Binding GUI: Extension procedures for adjusting a core page

If GUI changes are needed:

- Create subclasses of all the classes involved
  - View subclass:
    Implement required optical changes there, e. g. additional widgets
  - Binder subclass:
    - `getViewClass` has to be overwritten to return the new view subclass
    - Implement data binding for additional widgets
    - Implement required changes in the event handling
  - Page controller subclass:
    In `configureBinders`, the original binders are to be replaced by the new subclasses
- Add an entry to a customer-specific RemoteClassMapper:
  Map the original dialog page to the new page controller subclass

Exercise 2.2.1: GUI: Extending a table

New table view column

- On the "Search/Result" page, the results table should be extended to include the column "age" (“Alter”) in order to meet the customer's specific needs.

Exercise 2.2.2: GUI: Change in behaviour

New A3kHandler

- On the "Search/Criteria" a page, the extended search logic should be executed when the search button is clicked.

Exercise 2.2.3: GUI: Shift a table column

GUI changes: Change column order in a table

- The order of the columns "first name" and "surname" should be switched around in the results list.

2

2 Extension procedure (customer-specific adjustments)

2.1 Dialog layer

2.2 GUI layer

2.2.1 Visual aspects (views)

2.2.2 Data Binding

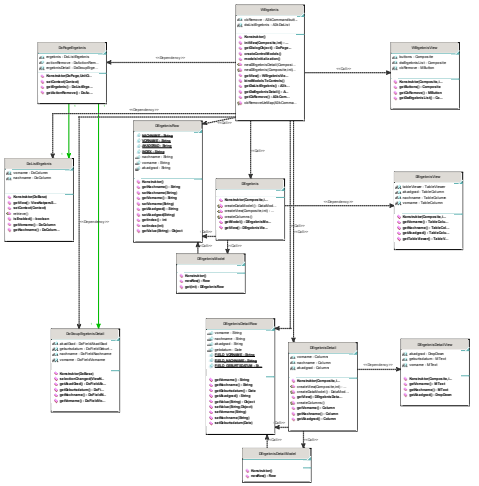
2.2.3 PowerBuilder emulation basics

2.2.4 PowerBuilder emulation extension

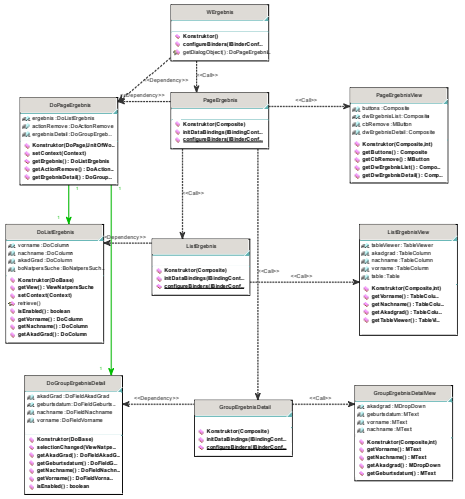
2.3 Internationalisation

Comparison of GUI layer structure

- PB GUI:
  Old GUI framework with PB emulation,
  since Core 8.8: Classes
  - Dialog
    + Mapping
    + Controller + Model + Row
    + Binding
    + View



- Data Binding GUI:
  New GUI framework with Data Binding,
  since Core 10.5: Classes
  - Dialog
    + Binding
    + View



Allianz

PowerBuilder emulation with “Mapping and Binding”

- GUI framework since core 8.8 (Java GUI migration):
  - Reproduction of the PowerBuilder framework
  - Former DataWindow split into controller, model, row classes
  - Identification of columns/fields via name strings
  - Boilerplate code, contains considerable repetition
  - Attention: Numerous sources of error
  - Code generator “MVC Wizard” available for practicable development
- Complex connections:
  - “Mapping” between PowerBuilder elements and dialog objects
  - “Binding” between PowerBuilder elements and SWT control elements

Exercise 2.2.4: PB GUI: Basic structure

Basic structure for PB GUI exercises

- Import the given plug-in `at.allianz.gfb.core.training.person.pb.gui`. It includes the same functionality as your existing plug-in `at.allianz.gfb.core.training.person.gui`, but is implemented with the PowerBuilder GUI emulation framework.

Differences for dialog objects and view objects (1)

Both framework versions have the same purpose:
to form the connection between view objects and dialog objects

Only small differences:

Data Binding

- Dialog class for buttons/menu items:
  - `DoCommand` without event handling: Event handling is in the GUI only
- Widget class for drop-down list boxes
  - `MDropDown`

PowerBuilder Emulation

- `DoAction` with execute method: A part of the event handling is in the dialog object, another part is in the GUI
- `DropDown`

Differences for dialog objects and view objects (2)

Data Binding

- Refresh behaviour:
  - All refreshes need to get triggered explicitly;
    hook method `DoPage.activate` can be useful for this
- How to perform a refresh:
  - Methods in the specific dialog objects:
    `retrieveValues()`,
    `retrieveEnabled()`,
    `retrieveVisible()` …

PowerBuilder Emulation

- Default setting `RETRIEVAL_MODE_ALWAYS` automatically refreshes on each page activation
- Class `FrontEndNotifier` can be used to request a refresh of the complete page:
  `notifyPageRetrieve()`,
  `notifyUpdateEnabled()`,
  `notifyUpdateVisibility()` …

PB GUI: The page controller

- A subclass of `WPage`
- Also called “GUI page”; PowerBuilder diction: “Window”
- Contains the controller logic for the whole page
- Positioned in the package `logic`

Responsibilities

- Creates the page view object
- Contains the PowerBuilder emulation elements:
  - Simple controls, like `A3kCommandbutton` or `A3kSle`
  - “DataWindows” (PB diction for groups or lists), like `A3kDatawindow` or `A3kDwList`, with DataWindow controller attached
- Defines their connections:
  - “Mapping” connects them with dialog objects
  - “Binding” connects them with view components
- Contains event handling code

PB GUI: Responsibilities of the page controller (1)

Creation of the presentation object

- `initView` instantiates the page view object

```
@Override
public WExampleView initView(Composite parent, int style) {
    return new WExampleView(parent, style);
}
```

PB GUI: Responsibilities of the page controller (2)

Helper methods for Binding and Mapping

- `getView` returns the view object in a typed way

```
@Override
public WExampleView getView() {
    return (WExampleView) super.getView();
}
```

- `getDialogObject` returns the dialog object in a typed way

```
@Override
public DoPageExample getDialogObject() {
    return (DoPageExample) super.provideDialogObject();
}
```

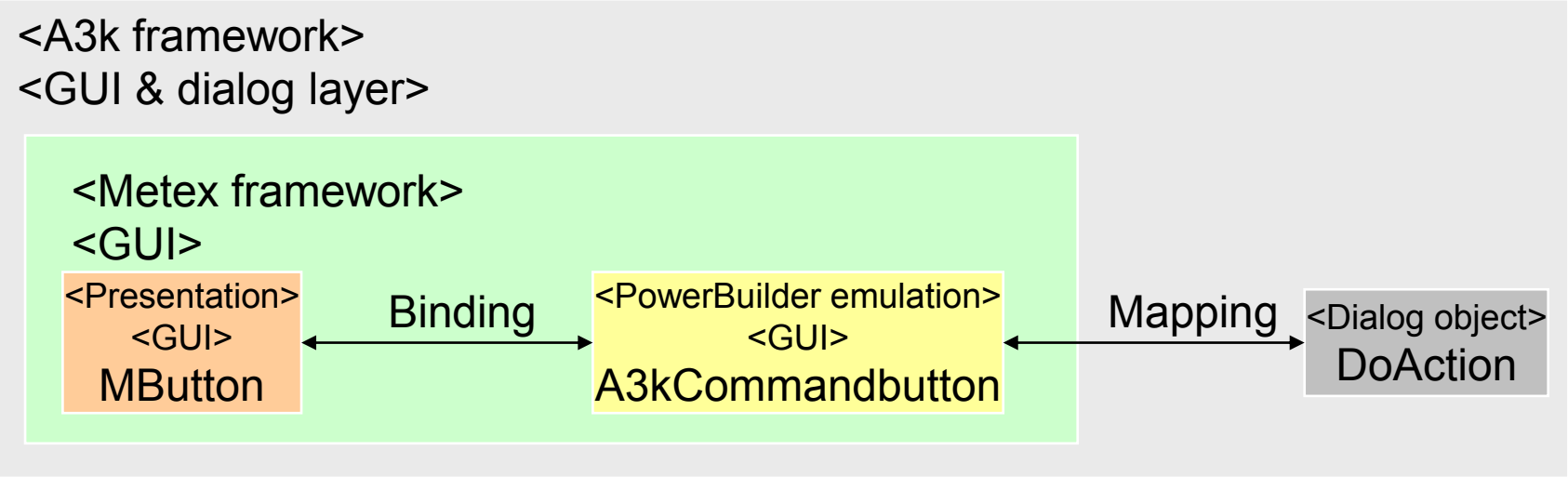
PB GUI: Mapping & Binding

- Mapping

  Association of the GUI controller component (PowerBuilder emulation) with the dialog logic component

- Binding

  Association of the GUI controller component (PowerBuilder emulation) with the GUI presentation component (SWT element)



PB GUI: Responsibilities of the page controller (3)

Simple PowerBuilder elements and Mapping

- `createControlModels` creates PB elements and performs their mapping
  - Instantiation of e. g. `A3kCommandButton`
  - Mapping to the dialog object in the overridden method `ue_map`

```
@Override
public void createControlModels() {

    cbSearch = new A3kCommandbutton() {

        @Override
        public void ue_map() {
            map(getDialogObject().getGroupCriteria().getSearch());
        }
    };

    …
```

  - Often the mapping gets delegated into a separate page controller method
  - Alternatively an event handler can contain the mapping code

PB GUI: Responsibilities of the page controller (4)

Simple PowerBuilder elements and user event handling

- `createControlModels` also defines event handling
  - Attach an event handler with `setEventHandler`, e. g. `A3kCommandButtonEventHandler`

```
@Override
public void createControlModels() {
    …

    cbSearch.setEventHandler( new A3kCommandButtonEventHandler() {

        @Override
        public void onClicked(ButtonModel button) {
            super.onClicked(button);
            akte.tabgoto(…); // some GUI activity
        }
}
```

  - Alternatively the method `ue_postclicked` can be overridden
  - Special event handler types available, depending on the element type

PB GUI: Responsibilities of the page controller (5)

Simple PowerBuilder elements and Binding

- `bindModelsToControls` binds the PB elements to SWT widgets
  - Binding: Call `addControl` for each widget

```
@Override
public void bindModelsToControls() {
    addControl(getView().getExampleButton(), cbExample);
    …
}
```

Allianz

PB GUI: DataWindow class structure

"DataWindow" is the PowerBuilder term for group and list components

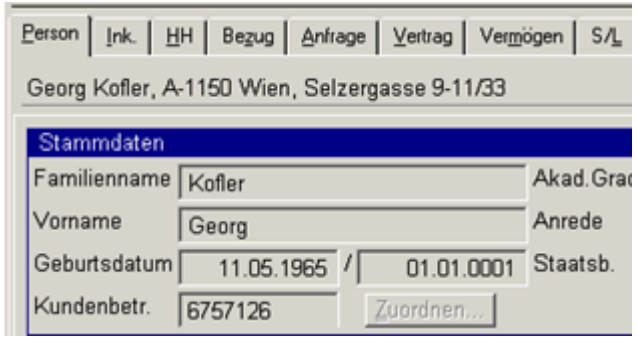
- A DataWindow consists of the following parts:
  - Controller
    Located in package `logic`
  - View
    Located in package `view`
  - Model and Row
    Located in package `model`
- Creation of the complete structure via code generator "MVC Wizard"

- View part normally needs custom implementation
- Controller can optionally contain special logic:
  - "Expressions" (formatting which depends on values)
  - "Computed columns" (additional computation logic)

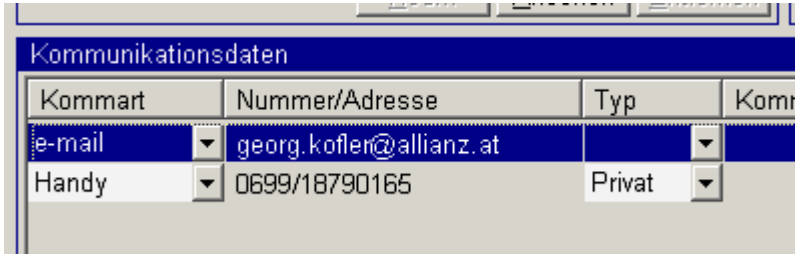
PB GUI: DataWindow types

Controller types (subclasses of `DataControl`):

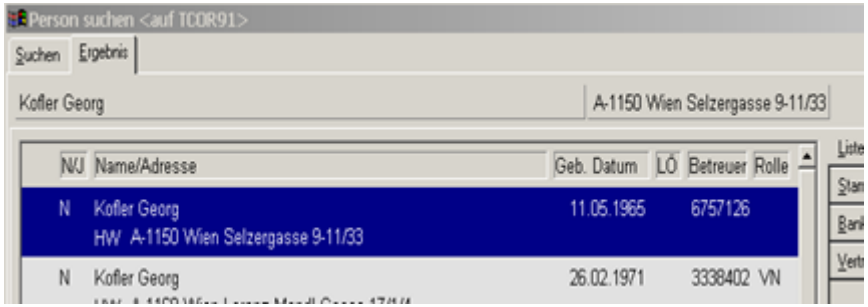
- `DataPanel`
  - Group component
  - Example: `DNatpersStamm`



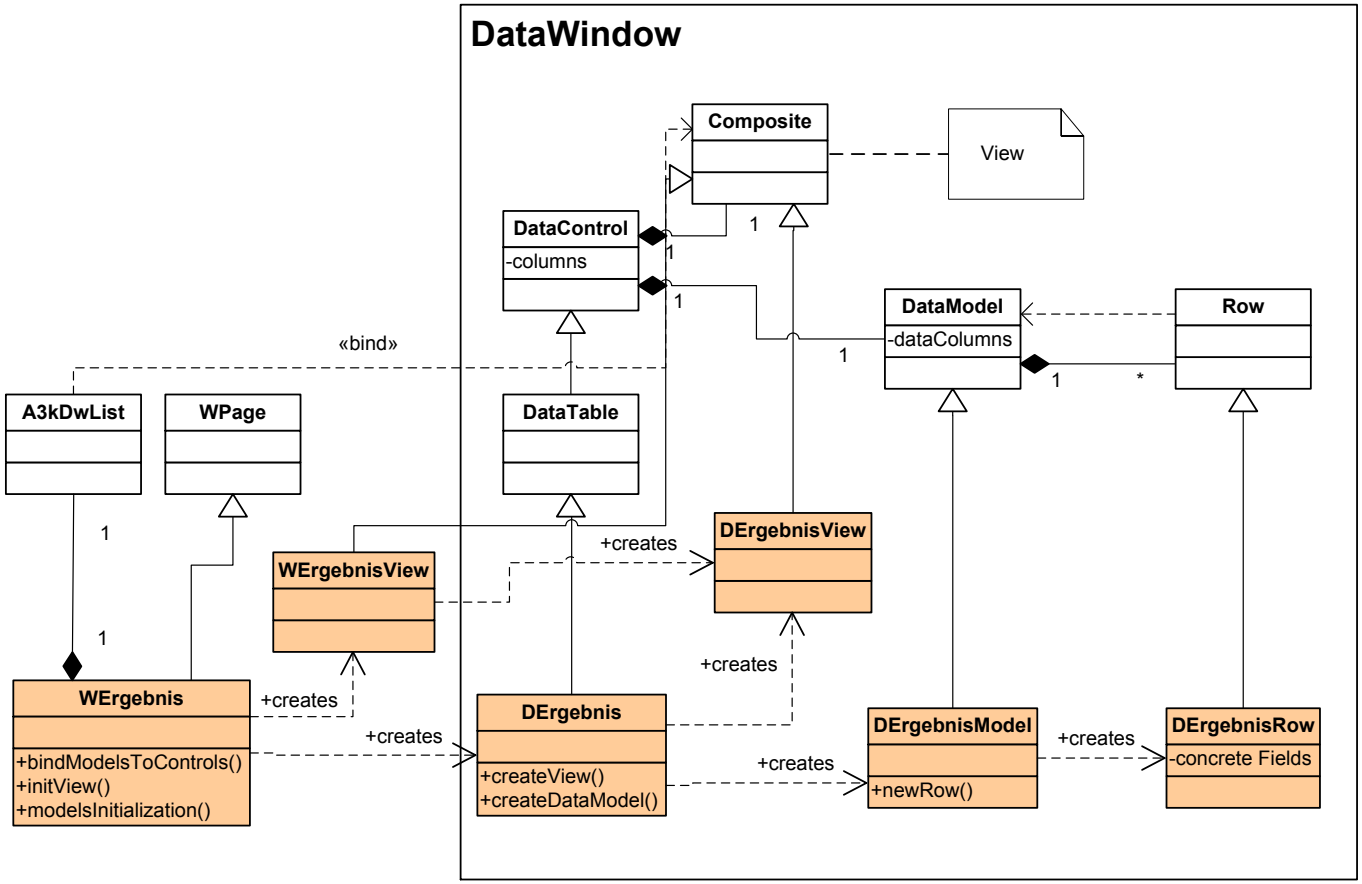
- `DataTable`
  - List with table presentation
  - Example: `DNatpersKommunikation`



- `DataForm`
  - List with multi-row list presentation
  - Example: `DSuchschirmPersonErgebnis`
  - Each row has it's own `RowView`



PB GUI: DataWindow responsibilities (1)



PB GUI: Responsibilities of the page controller (6)

DataWindow and Mapping

- `createControlModels` creates PB elements and maps them
  - Instantiation of e. g. `A3kDataWindow`
  - Mapping to the dialog object in the overridden method `ue_map`:
    - `map` for the complete component
    - `mapfield`, `mapdomainfield` for each field

```
dwExample = new A3kDatawindow() {
    @Override
    public void ue_map() {
        DoGroupExample doGroup = getDialogObject().getGroupExample();
        map(doGroup);
        mapfield(DExampleRow.FIELD_FIRSTNAME, doGroup.getFirstName());
        mapfield(DExampleRow.FIELD_SURNAME, doGroup.getSurname());
        mapdomainfield(DExampleRow.FIELD_ACADDEGREE, doGroup.getAcadDegree());
    }
};
```

PB GUI: Responsibilities of the page controller (7)

List DataWindow and Mapping

- `createControlModels` creates PB elements and maps them
  - Instantiation of `A3kDwList`
  - Mapping to the dialog object in the overridden method `ue_map`:
    - `map` for the complete component
    - `mapcolumn` or `mapdomaincolumn` for each column;
    - `mapindexcolumn` for the internal (invisible) index column

```
dwList = new A3kDwList() {
    @Override
    public void ue_map() {
        DoListExample doList = getDialogObject().getListExample();
        map(doList);
        mapcolumn(DListExampleRow.COLUMN_NAME, doList.getName());
        mapdomaincolumn(DListExampleRow.COLUMN_ACADDEGREE,
                doList.getAcadDegree());
        mapindexcolumn(DListExampleRow.COLUMN_INDEX);
    }
};
```

PB GUI: Responsibilities of the page controller (8)

DataWindow and event handling

- `createControlModels` also defines event handling
  - Event handling: Attach an event handler with `setEventHandler`, e. g. `A3kDwListEventHandler`

```
@Override
public void createControlModels() {
    …
    dwListResult.setEventHandler(new A3kDwListEventHandler() {

        @Override
        public void onRowFocusChanged(final DataWindow dataWindow,
                final int currentRow) {
            super.onRowFocusChanged(dataWindow, currentRow);
            // Do something ...
        }
    });
}
```

  - Alternatively methods of `A3kDataWindow`, `A3kDwList` can be overridden, e. g. `ue_postrowfocuschanged`

PB GUI: Responsibilities of the page controller (9)

DataWindow and Binding

- `bindModelsToControls` binds the PB elements to SWT widgets
  - Binding: Call `addControl` for the placeholder (`MGroup`/`Composite`)

```
@Override
public void bindModelsToControls() {
    addControl(getView().getExampleGroup(), dwExample);
    …
}
```

PB GUI: Responsibilities of the page controller (10)

DataWindow controller initialisation

- `modelsInitialisation` creates the DataWindow controllers
  - Instantiates the DataWindow controller
  - Assigns it to the PB element with `setDataControl`

```
@Override
public void modelsInitialization() {
    DExample dExample = new DExample(dwExample.getComposite(), SWT.NONE);
    dwExample.setDataControl(dExample);
}
```

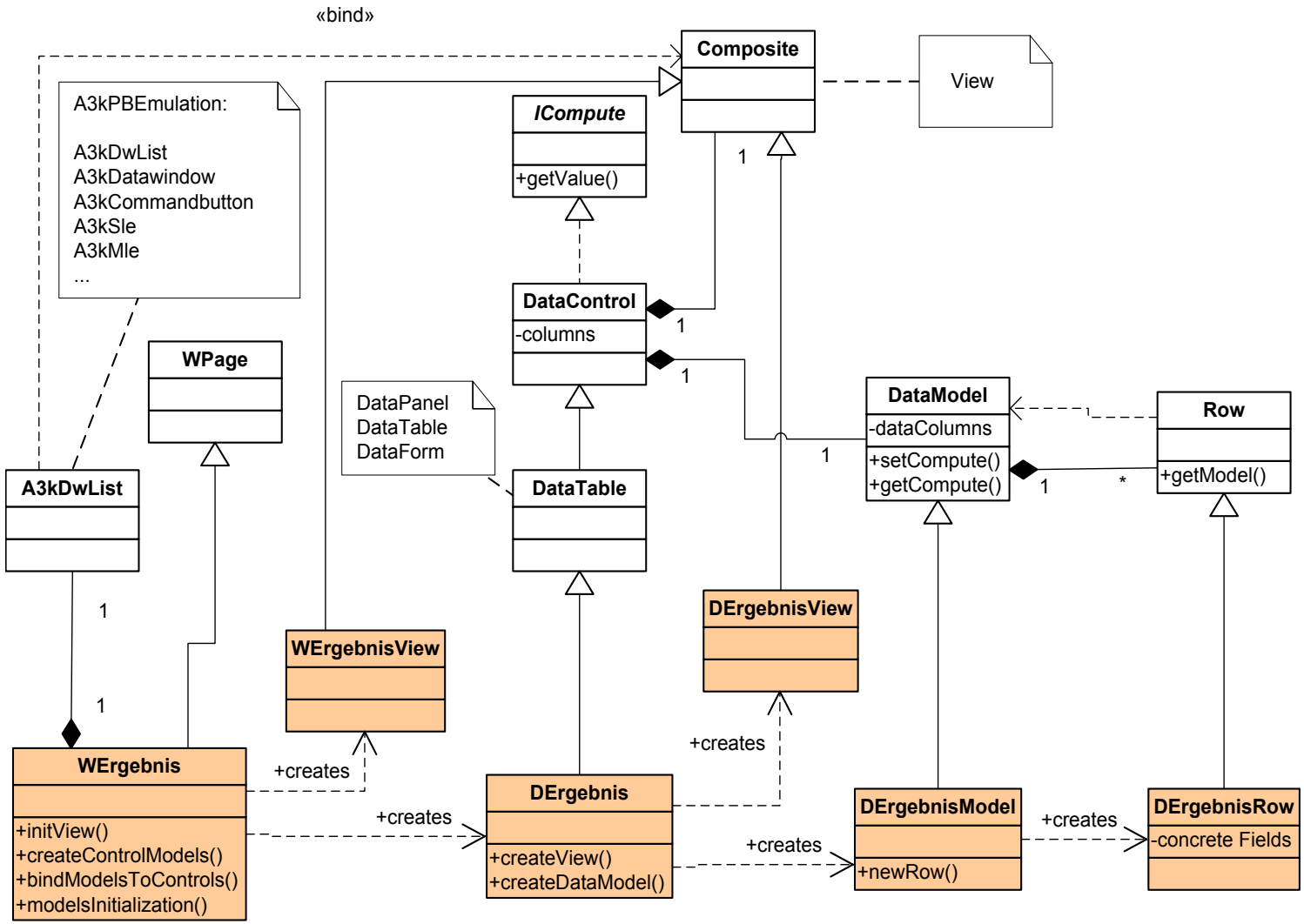
  - If the DataWindow view contains controls which need to be bound separately (e. g. `MButton`): Initialisation code can alternatively be put into `bindModelsToControls`

PB GUI: The page controller (summary)

Methods and their responsibilities

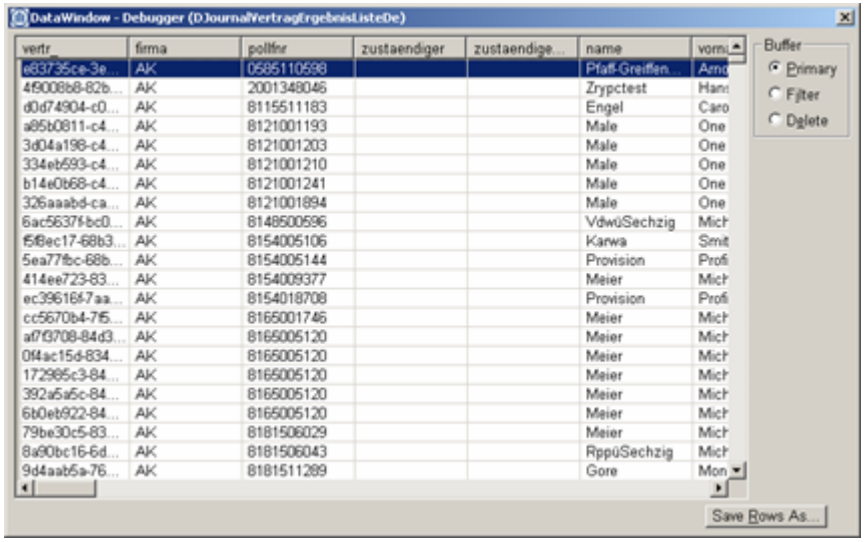
- `initView` creates the page view object
- `getView` and `getDialogObject` serve as helper methods
- `createControlModels` initialises the PowerBuilder emulation elements
  - Simple controls like `A3kCommandbutton` or `A3kSle`
  - DataWindows like `A3kDatawindow` or `A3kDwList`
- Connections:
  - Mapping to dialog objects in `ue_map` or in an event handler, `map`
  - Binding to SWT widgets in `bindModelsToControls`
- Event handling code e. g. in `ue_postclicked` or in an event handler, e. g. `postclicked`

PB GUI: Overview



PB GUI: Debug functions

- Ctrl + right click (only on DataWindow) – provides information on the data content



- Requirement for debug functions: command line parameter `/DEBUG`

Exercise 2.2.5: PB GUI: Create extension plug-in, adjust dialog layer

Plug-in for PB GUI extensions; dialog layer adjustments

- Import the partially prepared plug-in `de.allianz.abs.training.person.pb.gui` and analyse its code.

Allianz

2

2 Extension procedure (customer-specific adjustments)

2.1 Dialog layer

2.2 GUI layer

2.2.1 Visual aspects (views)

2.2.2 Data Binding

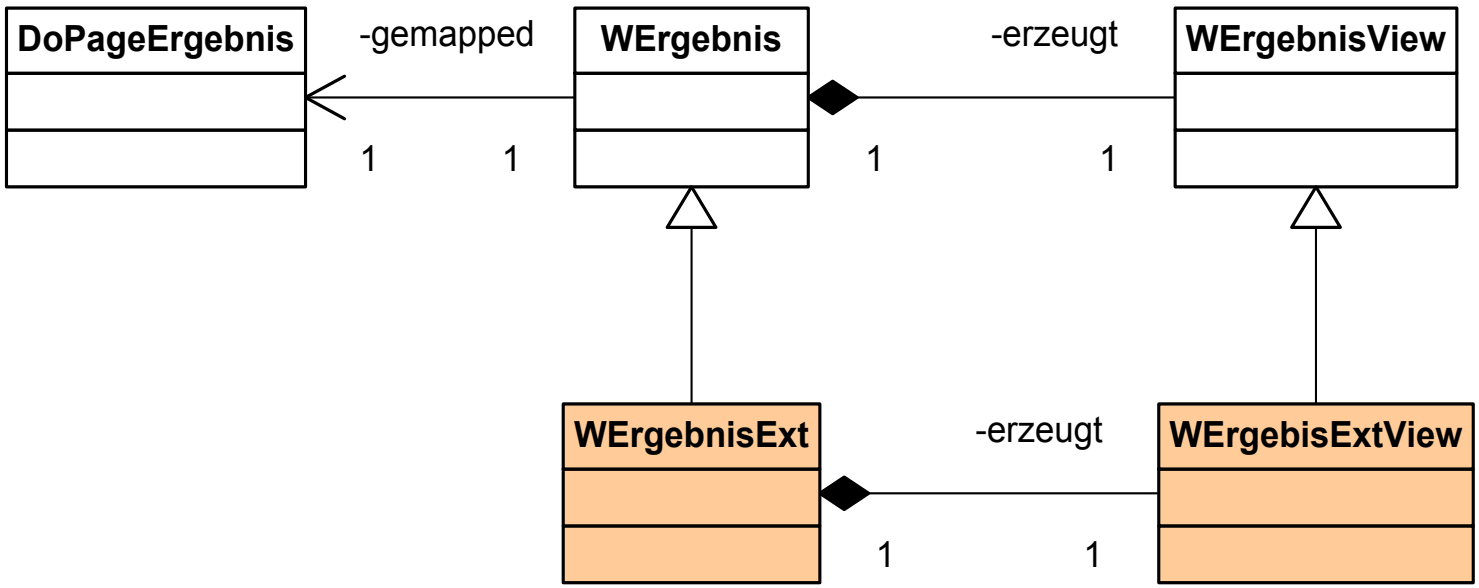
2.2.3 PowerBuilder emulation basics

2.2.4 PowerBuilder emulation extension

2.3 Internationalisation

PB GUI: Extending a page (1)

- Create page controller and page view successors



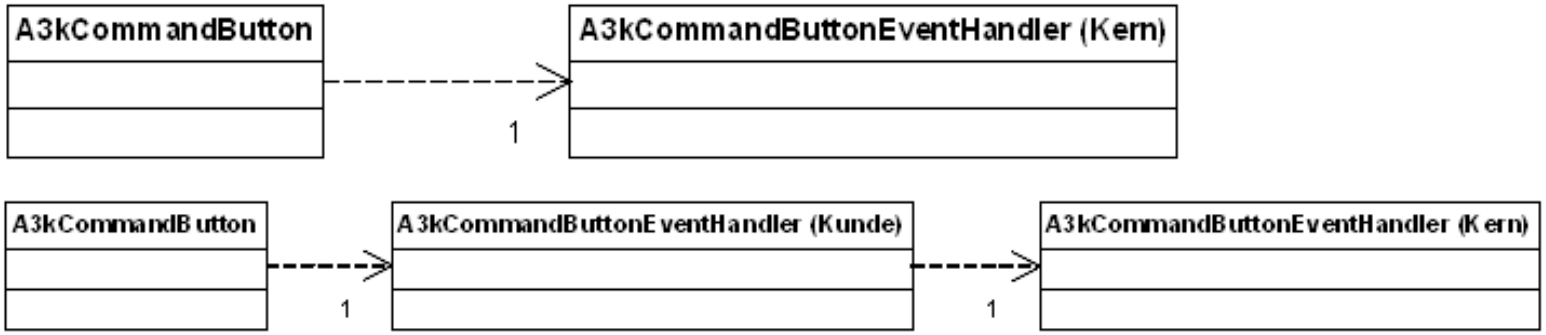
PB GUI: Extension procedures for adjusting a core page

If GUI changes are needed:

- Create subclasses of all the classes involved
  - View subclass:
    Implement required optical changes there, e. g. additional widgets
  - Page controller subclass:
    - Overwrite the view creation and use the extended view subclass
    - Implement mapping and binding for additional widgets
    - Implement required changes in the event handling
    - Use an extended DataWindow, if required
- Add an entry to a customer-specific RemoteClassMapper:
  Map the original dialog page to the new page controller subclass

PB GUI: Extending a page (2)

- Change in behaviour: Register additional EventHandler



- An EventHandler can be registered using the method setEventHandler
- Processing order:
  - Most recently registered EventHandler (chaining via super calls)
  - Event method (the default implementation for `ue_postclicked`)
  - Predecessor can be called using the `getEventHandler()` method

PB GUI: Extending a page (3)

Register an event handler in the method `createControlModels()`

- Customer code without core call

```
getMyButton().setEventHandler(
        new A3kCommandButtonEventHandler() {
    @Override
    public void postClicked(A3kCommandbutton button,
            Requestinfo requestInfo){
        … // Customer code without core logic
    }
});
```

- Customer code with core call

```
getMyButton().setEventHandler(
        new A3kCommandButtonEventHandler() {
    @Override
    public void postClicked(A3kCommandbutton button,
            Requestinfo requestInfo) {
        … // Customer code before the core logic
        super.postClicked(button, requestInfo);
        … // Customer code after the core logic
    }
});
```

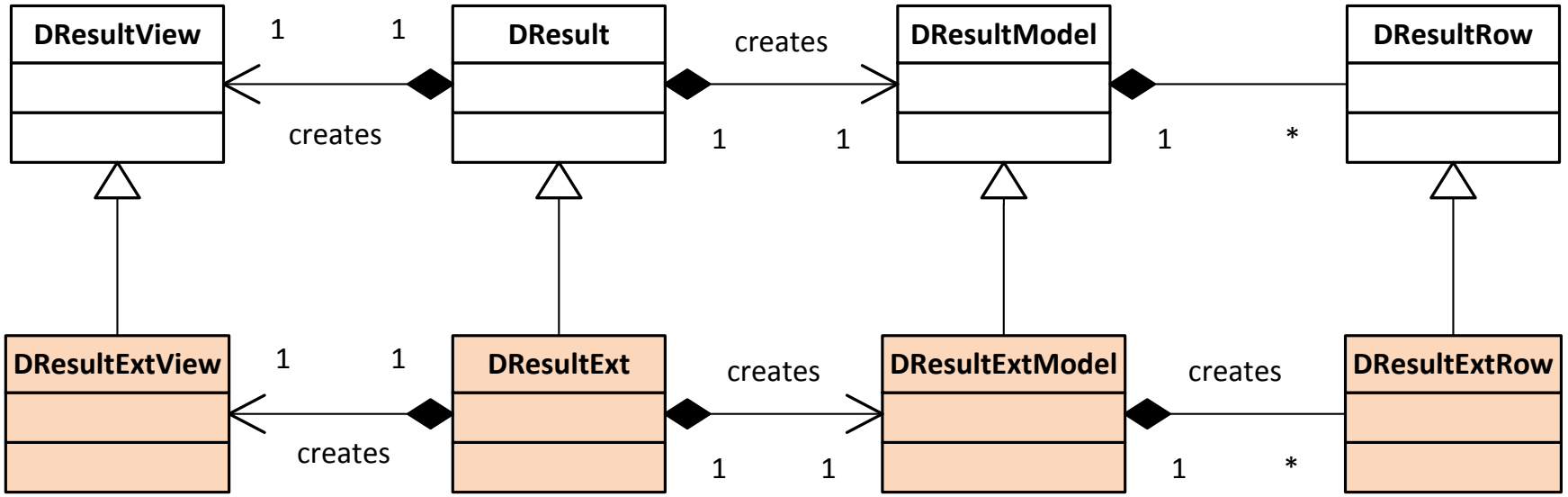
Allianz

PB GUI: Extension of a DataWindow (1)

- Achievable objectives
  - Add controls
  - Remove controls
  - Modify controls
- Procedure
  - Create derivation of DataWindow (not a copy!) → MVC Wizard
  - Replace original DataWindow with its successor:
    - Create page successor
    - Replace original page with its successor
    - Therein, generate successor DataWindow instead of the original one

PB GUI: Extension of a DataWindow (2)

- Create DataWindow successor



PB GUI: Extension of a DataWindow (3)

- Replace original DataWindow with its successor
  - Create and insert successor page
  - Map additional fields/columns
    - Overwrite a protected mapping method, if it exists, or use an EventHandler
  - Set DataControl

```
public void modelsInitialization() {
    super.modelsInitialization();
    getDwErgebnis().setDataControl(
            new DErgebnisErweiterung(getDwErgebnis().getComposite(), SWT.NONE));
}
```

Exercise 2.2.6: PB GUI: Extensions in the old fashion

GUI extensions with the PowerBuilder GUI emulation framework

- Redo all GUI exercises you already did with the Data Binding framework with the PowerBuilder emulation framework.

Quiz 4

Quiz 4 – GUI Layer Extension Concepts (1)

What are the objectives when extending the GUI layer?

a. To modify controls
b. To remove controls in exceptional cases
c. To call different dialog logic
d. To change event handling

When removing controls, what is true?

a. It is better to use the dialog layer to do so, if possible
b. A customer must also remove the business logic for the control
c. When using GridLayout, we can use the layout attribute “exclude”
d. A control is removed automatically when the business logic gets deleted

To change a GUI view, which classes have to be inherited when using the new data binding?

a. The View class
b. The Binder class
c. The Persistent class
d. The page controller class

What are some of the drawbacks of the PowerBuilder emulation?

a. Code is very complex and unintuitive
b. Code is not type-safe
c. Code won’t compile any more in core 15.5 and later versions
d. There is no documentation whatsoever about the emulation

Solution: a, b, c, d – a, c – a, b, d – a, b

Allianz

2

Extensions – internationalisation

- Objective: Replacement of texts used in the core system (resources)
- Solution: Additional properties files in a plugin fragment
  - Text adjustments are to be made using this method
  - No need to derive view classes for this purpose
  - Language with country extension “de_AT”/”de_DE” should be used (file name: messages_de_AT.properties, messages_de_DE.properties)
- Procedural steps with the SWT designer:
  - Open design view of the core view class
  - Create “new locale” using “Externalize strings”
  - Do not copy strings (“Copy strings from none”)
  - Put the properties file that is created into the right fragment

Exercise 2.3.1: Change user interface text using fragment

Core text change via fragment

- User interface texts of the core GUI, i.e. the original search criteria and results page, are to be changed.
  - Change GUI control element labels.
  - Change the tab/side-tab labels.
  - Change the validation error message.

Allianz

3

Allianz

3

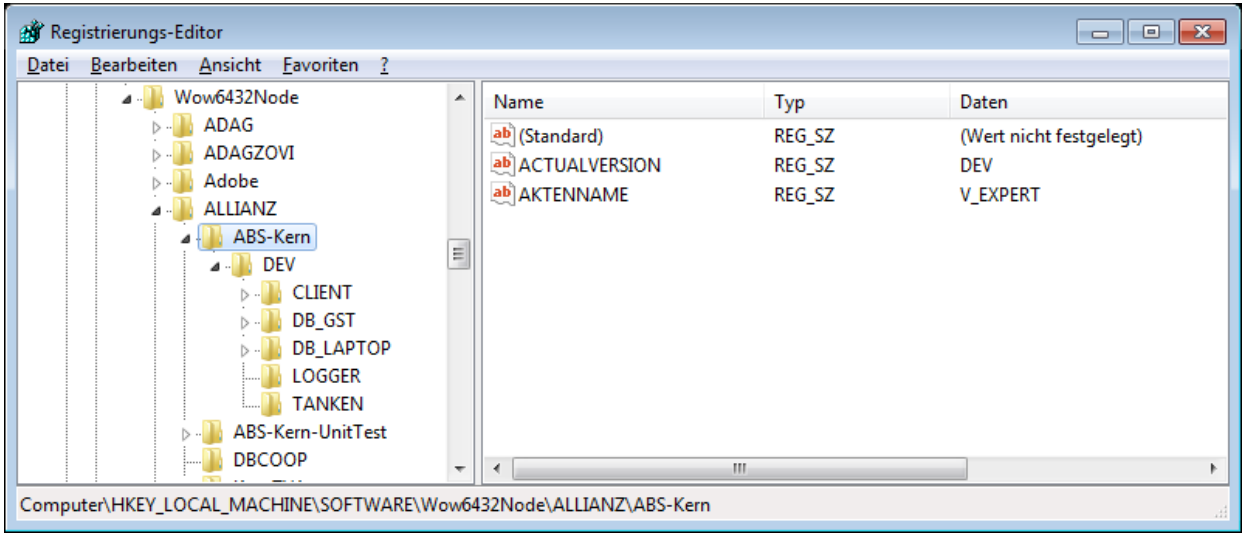
Configuration

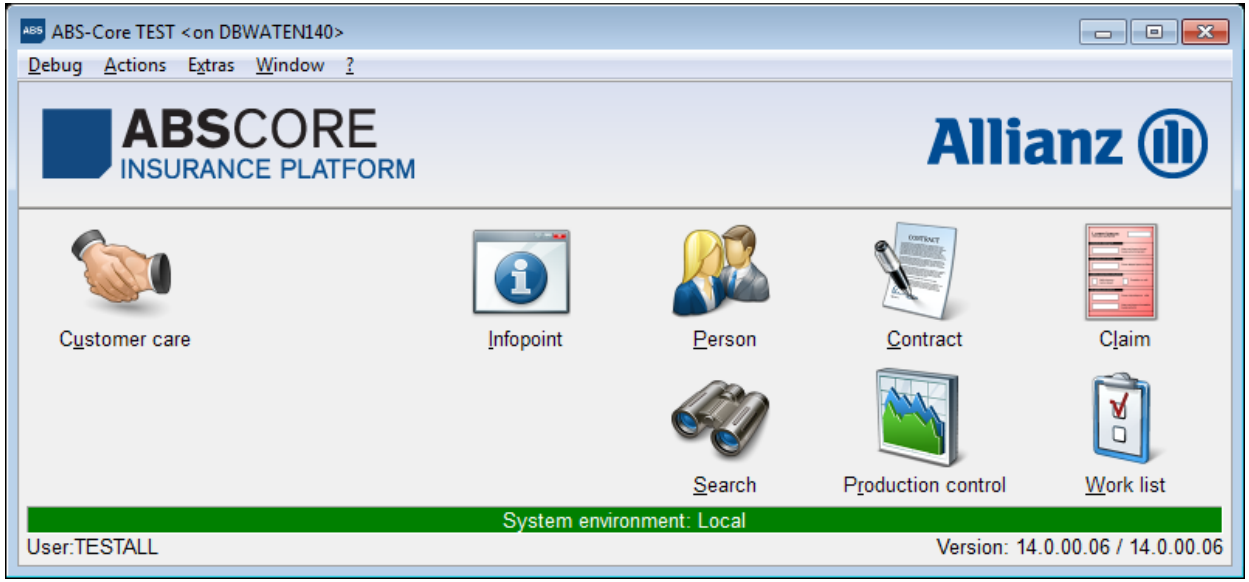
- Implementation is exchangeable
  - Extension point `at.allianz.a3k.applicationConfiguration`, Interface `IA3kApplicationConfiguration`, based on JNDI
    - Standard implementation (Core, AEV): uses Windows Registry
    - Customer-specific AZ D implementation: Uses cascaded `config.data` files
- Structure of configuration entries
  - <key offset>\<company>\<app name>\<version>
  - Can be defined via `IA3kApplicationConfiguration`
  - Separator is backslash “\”

Class Configuration

- Request configuration entries: `at.allianz.a3k.core.Configuration`
  - `getGlobalProps()` : `IA3kPropertyProvider`
    - Root of entire property tree
  - `getCompanyProps()` : `IA3kPropertyProvider`
    - Sub-tree <Offset>\<Company>
  - `getAppProps()` : `IA3kPropertyProvider`
    - Sub-tree <Offset>\<Company>\<App>
  - `getAppVersionProps()` : `IA3kPropertyProvider`
    - Sub-tree <Offset>\<Company>\<App>\<Version>
  - Key parts are from `IA3kApplicationConfiguration`

Example: Application name = ABS-Kern





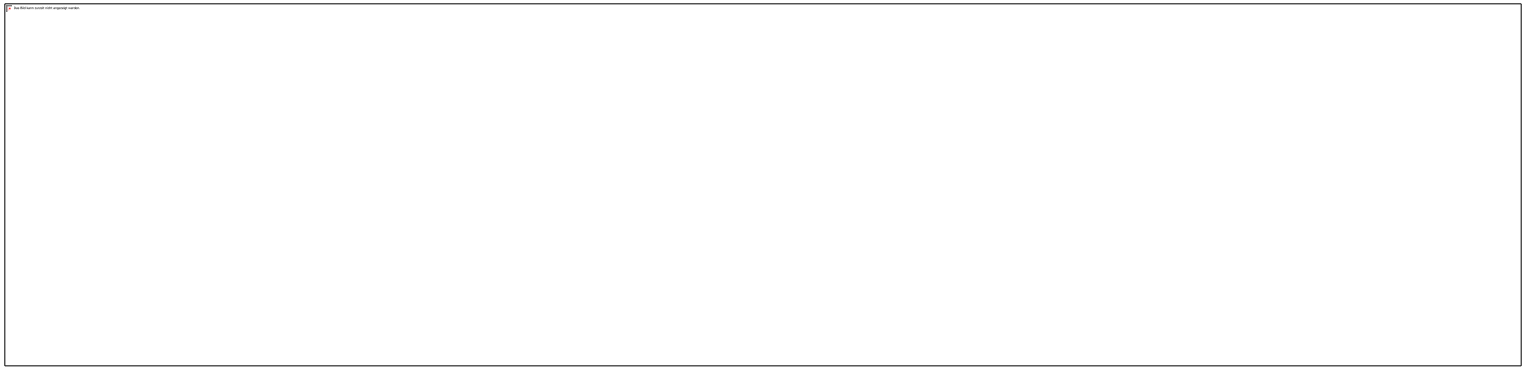
Allianz

3

Logging (1)

- Two loggers are available:
  - Global logger, can be identified using the following static call:
    - `A3kSession.getGlobalLogger()`
  - Session logger, has to be requested from a session instance:
    - `session.getLogger()`
- All loggers are based on the Java logging API (see http://docs.oracle.com/javase/7/docs/technotes/guides/logging/)
  - Example: `session.getLogger().info("Beispielmeldung");`

Logging (2)



- Each session has its own logger.
- The global logger is the parent of all loggers.
- Session loggers do not report to the global logger.
- Loggers can be configured using `LoggerConfigurator`. The framework makes a default implementation of this class available (`DefaultLoggerConfigurator`).

Logging (3)

- Base configuration
  - Console handlers and file handlers are attached to both loggers.
  - The ColumnBasedFormatter is attached to the session logger handler. ColumnBasedFormatter produces a row with the following columns for each log entry:
    - Version, application name, log level, log message, user ID, session ID
  - New columns can be defined as follows:
    - `session.getTypedAdapter(SessionLogAdapter.class).getFormatter().addColumn(column)`
  - Further configurations can be registered via the `"at.allianz.a3k.loggerConfiguration"` extension point

Logging (4)

- Setting the console handler
  - You can use the configuration key <App>.<Actual Version>.LOGGER.CONSOLE_LOGLEVEL to set the log level of the console handler.
  - Default value is OFF, i.e. if the key does not exist, the console handler is deactivated.
- Setting the file handler
  - You can use the configuration key <App>.<Actual Version>.LOGGER.FILE_LOGLEVEL to set the log level of the file handler.
  - Default value is OFF, like with the console handler.
  - The file that collects the log information can be determined using the configuration key <App>.<Actual Version>.LOGGER.FILENAME. If the key is not set, the file handler is deactivated, even if FILE_LOGLEVEL has been set.

Quiz 5

Quiz 5 – Cross-Functional Topics

Where can application configuration data be stored?

a. In the Windows registry
b. On a database server
c. In config.data files
d. In a cloud

What types of loggers are available in ABS?

a. A global logger
b. A java logger
c. A session logger
d. A VM logger

What properties do you need to configure in order to enable console logging?

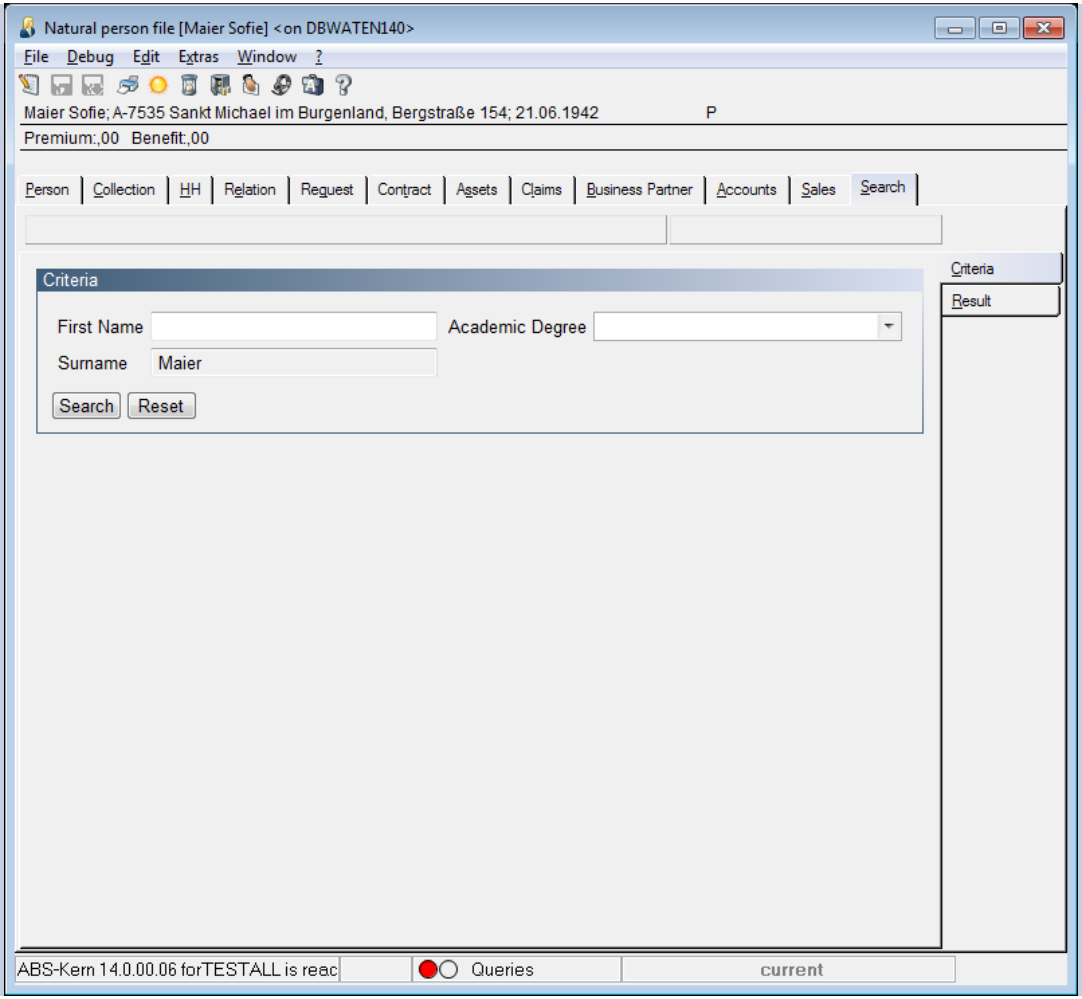
a. LOGGER.CONSOLE_LOGLEVEL
b. LOGGER.CONSOLE_LOG
c. LOGGER.CONSOLE_ENABLELOG
d. LOGGER.CONSOLE_LOGENABLE

What properties do you need to configure in order to enable file logging?

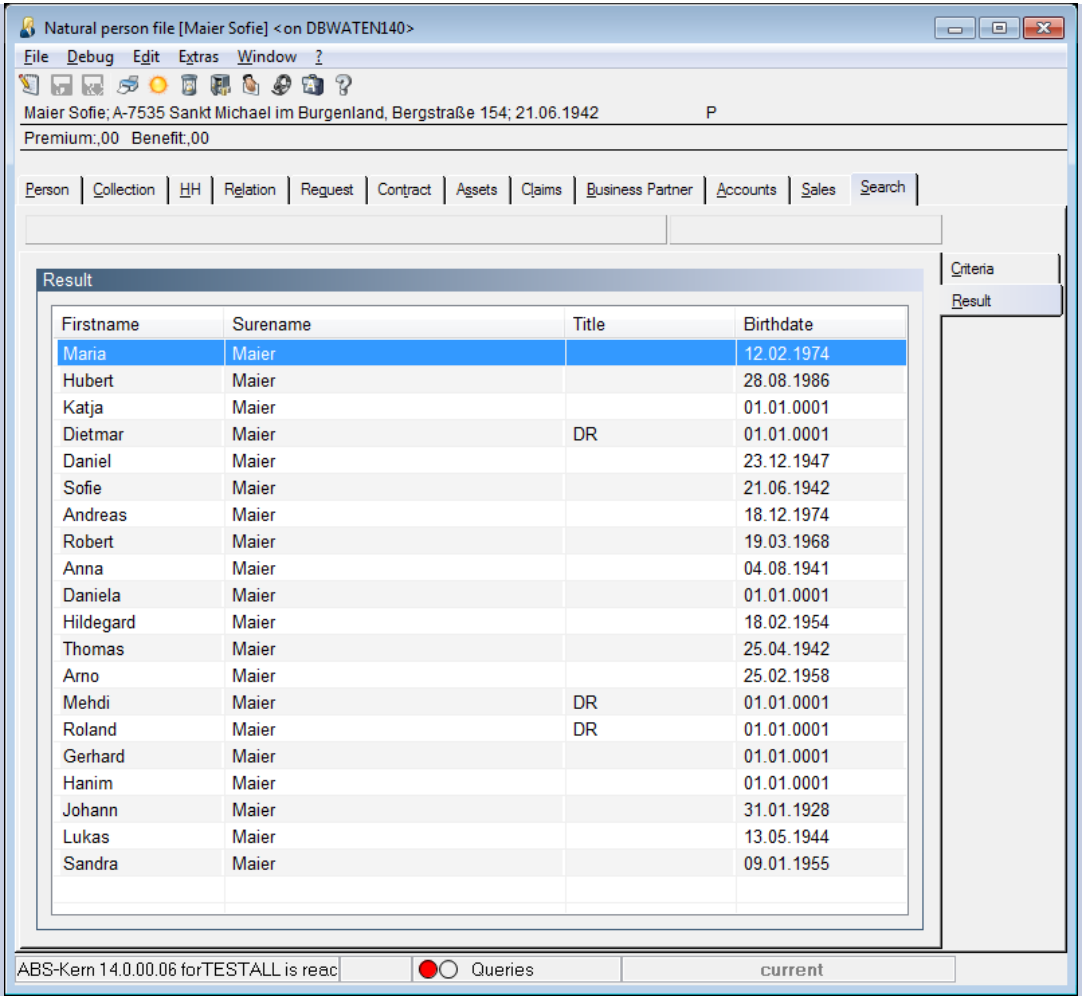
a. LOGGER.FILE_LOGLEVEL
b. LOGGER.FILE_LOG
c. LOGGER.FILE_ENABLELOG
d. LOGGER. FILENAME

Solution: a, c – a, c – a – a, d

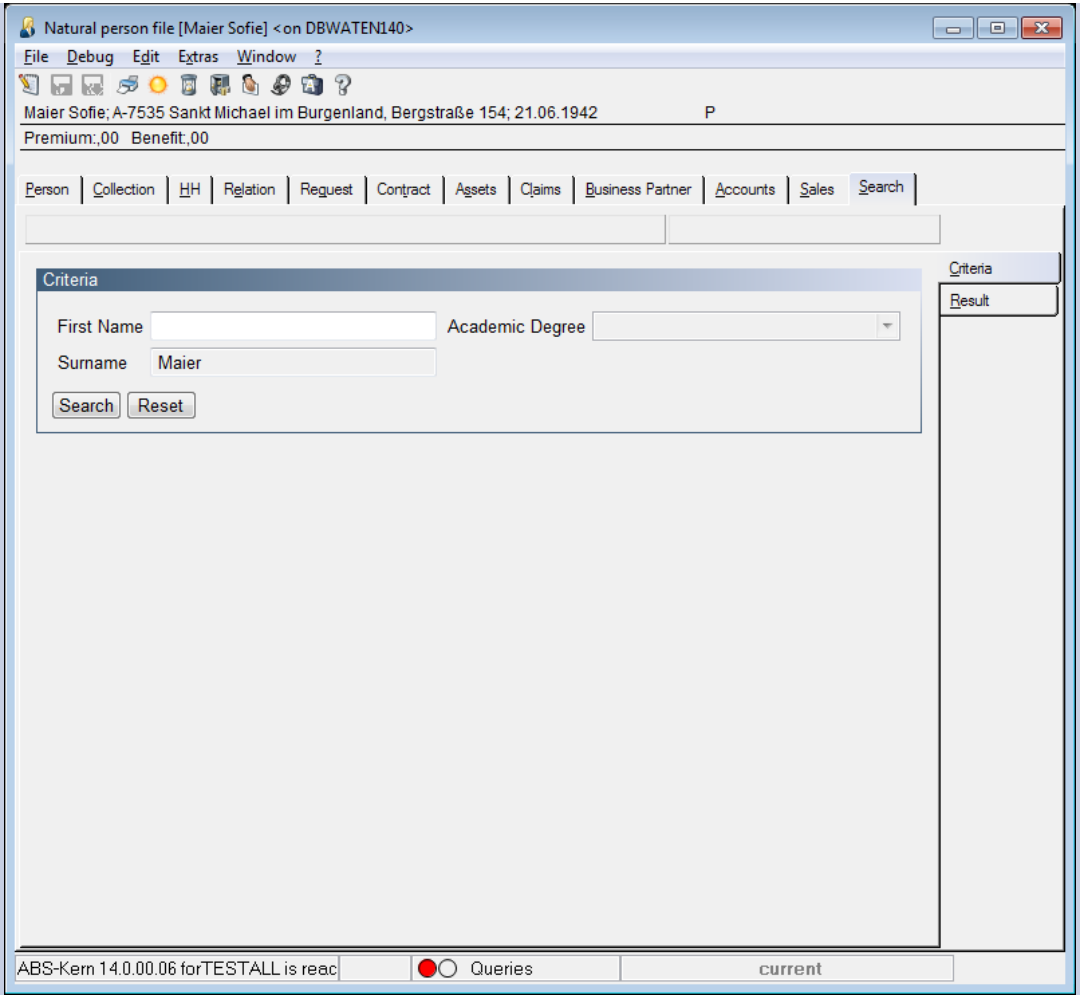
Exercises base concepts (chapter 1): First page ‘Criteria’



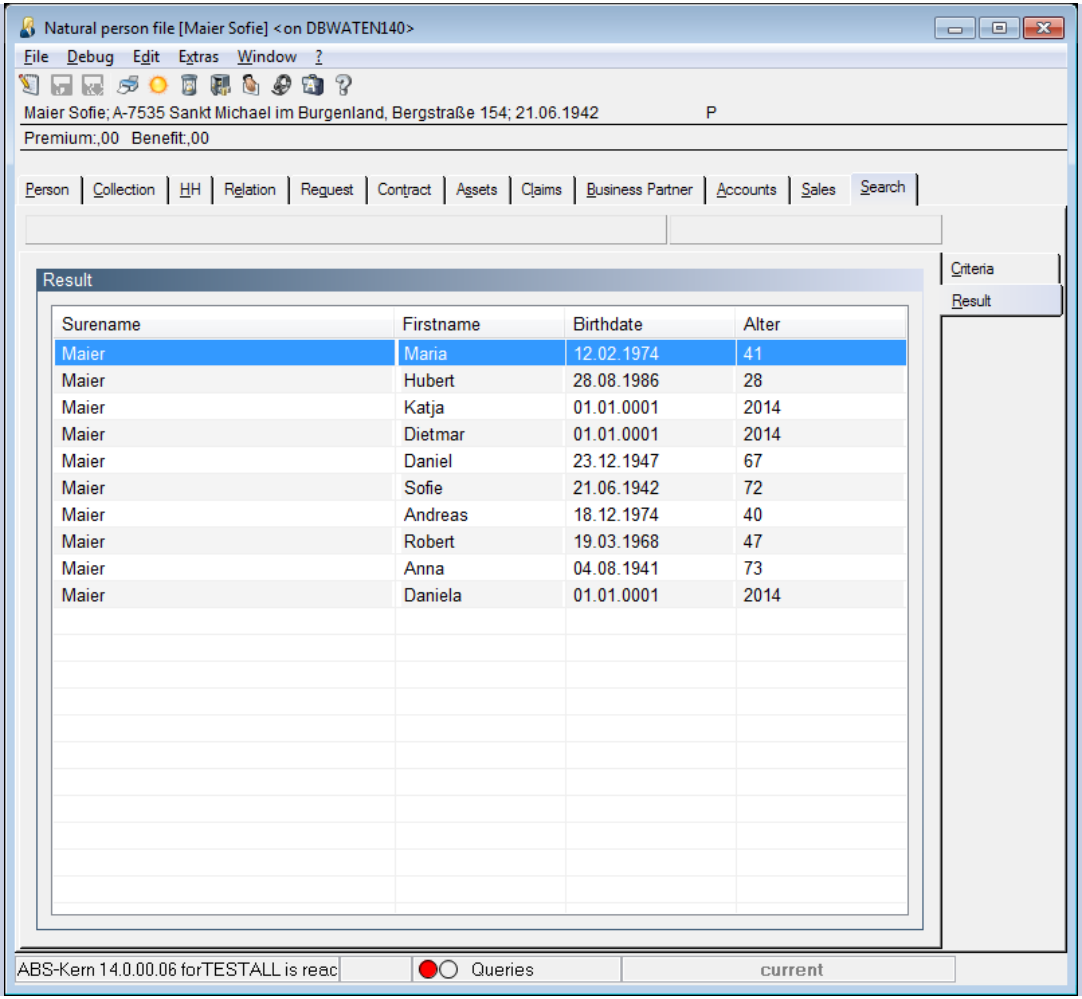
Exercises base concepts (chapter 1): Second page ‘Results’



Exercises extension concepts (chapter 2): First page ‘Criteria’, customised



Exercises extension concepts (chapter 2): Second page ‘Results’, customised



Allianz

metafinanz

we ensure **success**

*