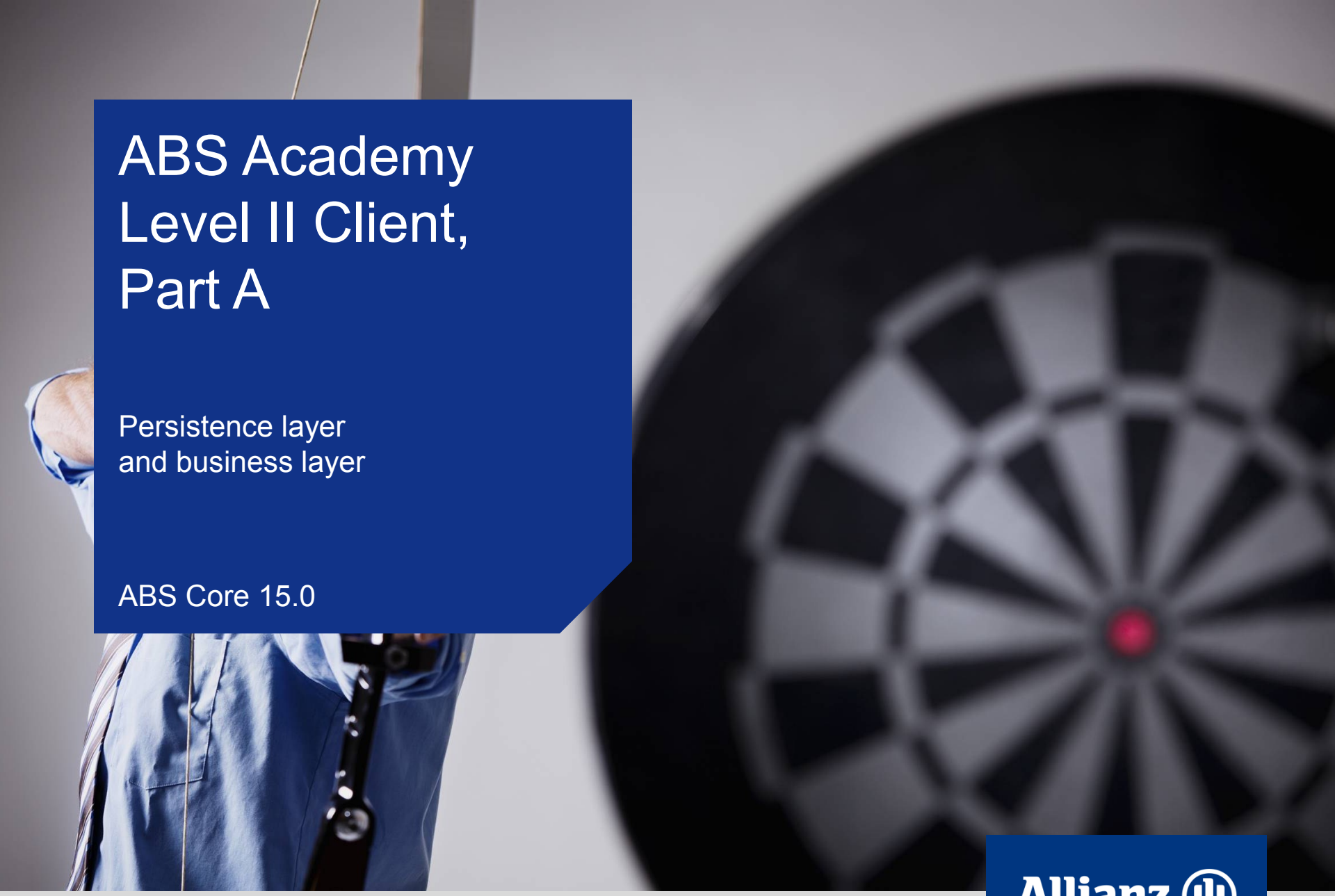


ABS Academy Level II Client, Part A

Persistence layer
and business 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 A

1 ABS/A3k overview
- 1.1 Introduction to ABS and history
- 1.2 A3k architecture overview
- 1.3 A3k documentation

2 Basic concepts (core development)
- 2.1 Persistence layer, DataContainer
- 2.2 Business logic layer

3 Extension procedure (customer-specific adjustments)
- 3.1 Persistence layer, DataContainer
- 3.2 Business logic layer

Allianz

1

Introduction to ABS

- ABS stands for **Allianz Business System**
- Standard software for insurances, for basically all business processes of an insurance enterprise
- Supports an operating model which unites insurance branches
- One of the Core Insurance Platforms of Allianz
- Development: AMOS Austria
- Core system which can be customized and extended
- Customers using ABS:
  - Allianz Group:
    Allianz Elementar (AEV, Austria), Allianz Suisse, Allianz Deutschland (AZ D, Germany), Allianz France, ABSi (Germany and France), Allianz Global Assistance (AGA, many countries), Allianz Greece
  - Others: ARAG Österreich, Tiroler Versicherung
  - Several projects for other customers in preparation

Allianz

GFB and ABS: History and status quo (1)

- From 1993 onwards: APES Framework and GFB
  - New application GFB (“Geschäftsfallbearbeitung”, business transaction handling) at Allianz Elementar
- Technology:
  - Client/server architecture
  - Client application based on PowerBuilder (4GL)
  - Database system: DB2 (mainframe)

Allianz

GFB and ABS: History and status quo (2)

- From 1998 onwards: A2k framework (APES 2000)
  - GFB as the new central insurance application of AEV
  - Replacement for older systems (shortly before Y2k)
  - Basis for GFB on the Internet
  - Creation of a company-wide data model
- Technology:
  - Clearer layer separation
  - GUI and business logic: PowerBuilder
  - Persistence layer ORBIT (C++, own development)
  - Database system: DB2 (mainframe) and Sybase (regional offices, notebooks)

Allianz

GFB and ABS: History and status quo (3)

- 2004 – 2007: New architecture A3k
  - Introduction of a complete new architecture on state of the art technology
  - Early 2006: Allianz Deutschland AG (AZ D, Allianz Germany) chooses A3k as the basis for its new central insurance application ABS (Allianz Business System)
  - Start of migration of GFB to A3k
  - September 2006: First version of the ABS core based on A3k
  - September 2007: ABS 1.0 goes into production at AZD
- Technology:
  - Main focus: Modular structure, configurability, extensibility
  - Separation of core and customer-specific features
  - New technology: Eclipse platform, Java 5
  - GUI still PowerBuilder at this time

Allianz

GFB and ABS: History and status quo (4)

- Since 2007: Revised architecture A3k plus
  - Reuse of A2k business logic leads to project acceleration
  - Gradual migration; avoidance of double maintenance
  - September 2007: First version of A3k plus; GFB on the new A3k plus basis goes into production at AEV
  - August 2008: ABS 2.0 (on A3k plus) goes into production at AZ D
- Technology:
  - Coexistence of A3k and A2k
  - A3k API for customers
  - Encapsulation of business logic and migration of the dialog logic
  - Persistence layer (ORBIT) ported to Java (JEE compliant)

Allianz

GFB and ABS: History and status quo (5)

- 2008 – 2010: Migration to Java GUI
  - Replacement for outdated PowerBuilder technology
  - From November 2008 onwards: Migration of the GUI layer from PowerBuilder to Java/SWT
  - Ergonomic improvements: adjustable font size and dynamic window layout
- Technology:
  - SWT/JFace as GUI technology, replaces PowerBuilder
  - GUI code translated from PowerBuilder to Java (by the company Metex)
  - Almost unchanged architecture

Allianz

GFB and ABS: History and status quo (6)

- 2010 – 2015: Framework improvements
  - Core 10.5:
    - Reworked GUI framework (Data Binding), simplifies GUI development
    - Introduction of a web framework, as a new part of the ABS core
  - Technology migration project:
    - Migration of the A2k business logic to A3k
    - Migration of the GUI to the new Data Binding framework (ongoing)
- Technology:
  - GUI framework based on Eclipse Data Binding
  - Web framework based on JSF(JBoss RichFaces), Spring Web Flow

Allianz

GFB and ABS: History and status quo (7)

- Present and future: Continuous advancements
  - Core 15.0/15.5:
    'UI Evolution' project: Introduction of the reworked user interface 'Aureus'
  - Core Insurance Service Layer (CISL):
    Unified service layer for all Core Insurance Platforms of Allianz
  - Ongoing improvements and innovation in several areas
- Technology:
  - Eclipse 4.x
  - Java 7/8

1

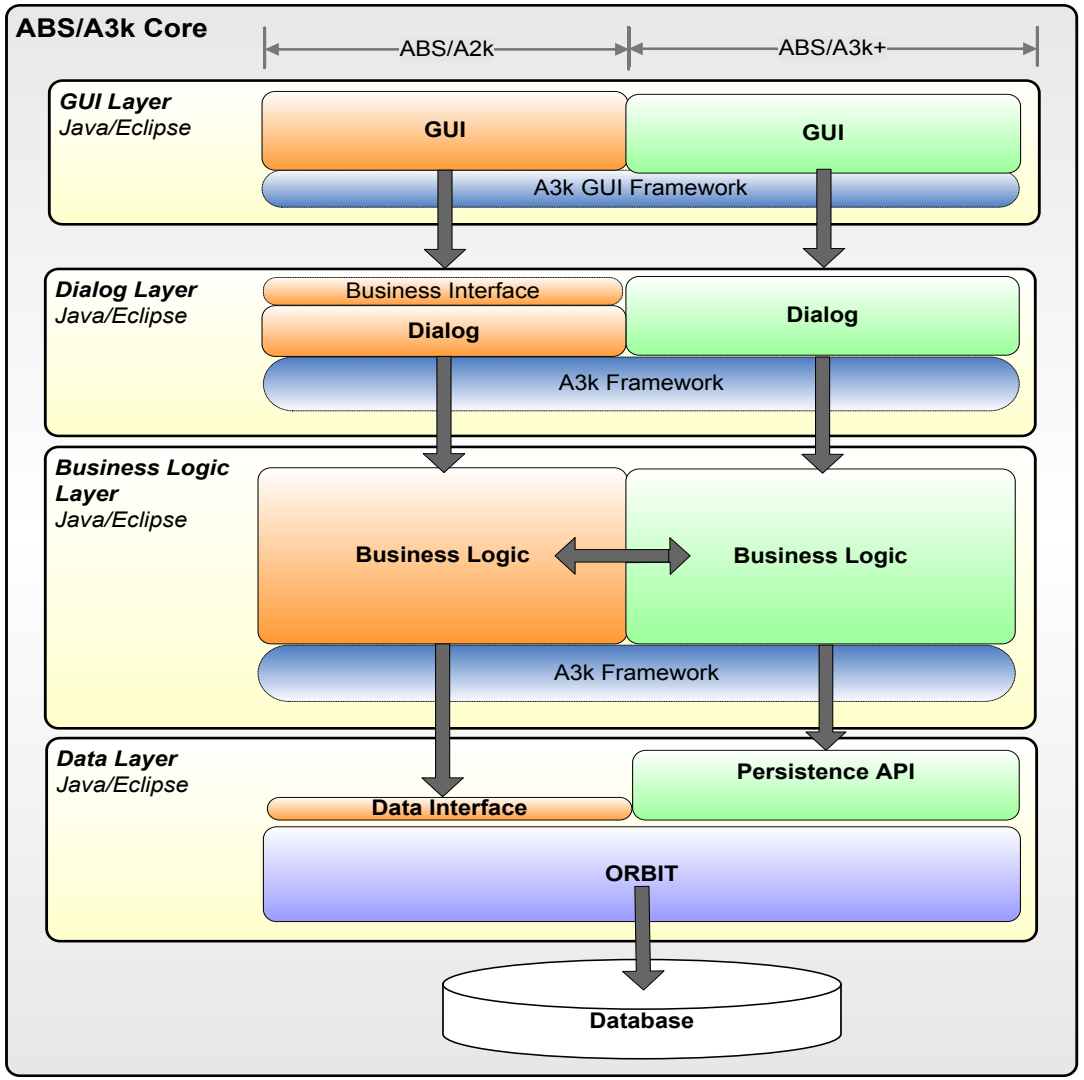
Allianz

Technological basis: Basic technologies behind A3k

- Java 7
  - As a contemporary programming platform
- Eclipse 4.3:
  - Bundle concept (OSGi, Eclipse Equinox), bundles alias plug-ins: enables the formation of components and thus modularisation
  - Extension point mechanism allows extension points to be offered in the core and customer-specific extensions to be added
  - Adapter concept to complement the API without inheritance and as an additional extension mechanism
- SWT/JFace
  - As a modern GUI technology

Allianz

Overview of the architecture



Allianz

Data layer

- Generated persistent object interfaces and classes
  - Interfaces for the persistent objects and Orbit implementation
  - PM classes (`PMPerson.java`, etc)
  - Domain constant interfaces
- Views
  - Dynamic views for database queries
  - Outdated: Static views `(*.view)` – used to generate classes
- Orbit schema files
  - `gfb_schema_*.xml` (OR mapping)
  - `table_schema_*.xml` (data model)
- Search engine

Allianz

Business logic layer

- Contains the functional logic
- Is independent of the presentation
  - Gets used by the rich client
  - Gets used by the web portal
- Makes use of the persistence layer

Allianz

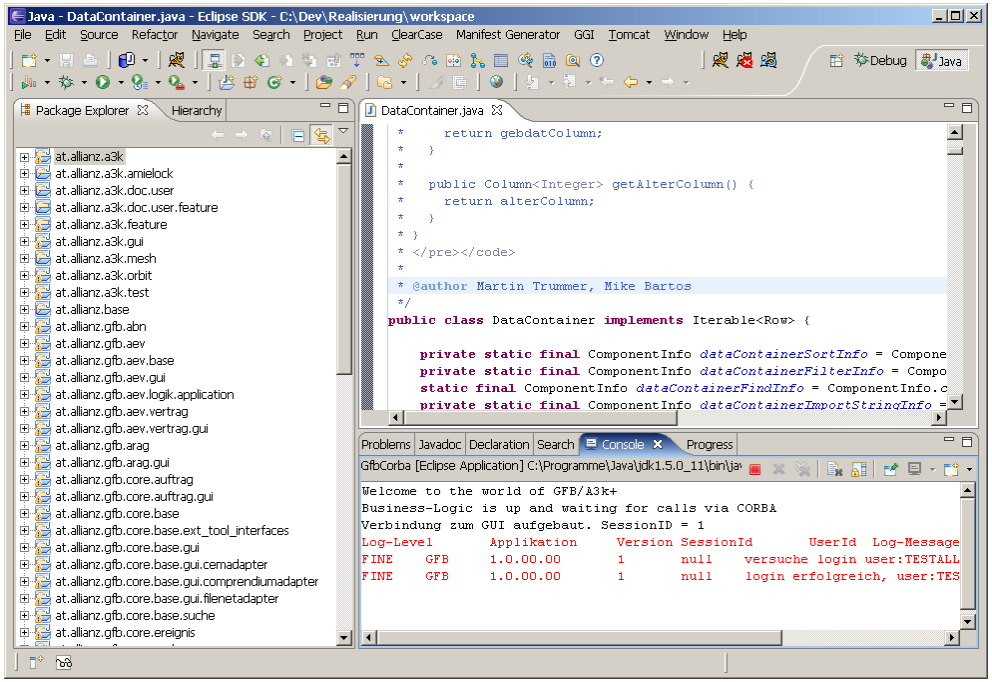
Dialog layer and GUI layer

- Dialog layer
  - Business logic façade
  - Groups the business logic into pages, groups, input fields …
  - Has the same logical component structure as the GUI
- GUI layer
  - Graphical user interface, built with SWT/JFace
  - Connection to the dialog layer

Allianz

Development infrastructure

- Eclipse
- Java
- A3k tools:
  - Voyager (debugger for ORBIT)
  - Persistence generator (includes domain generator)
- Code quality analysis tools:
  - ABSCodeAnalyzer, FindBugs, etc.

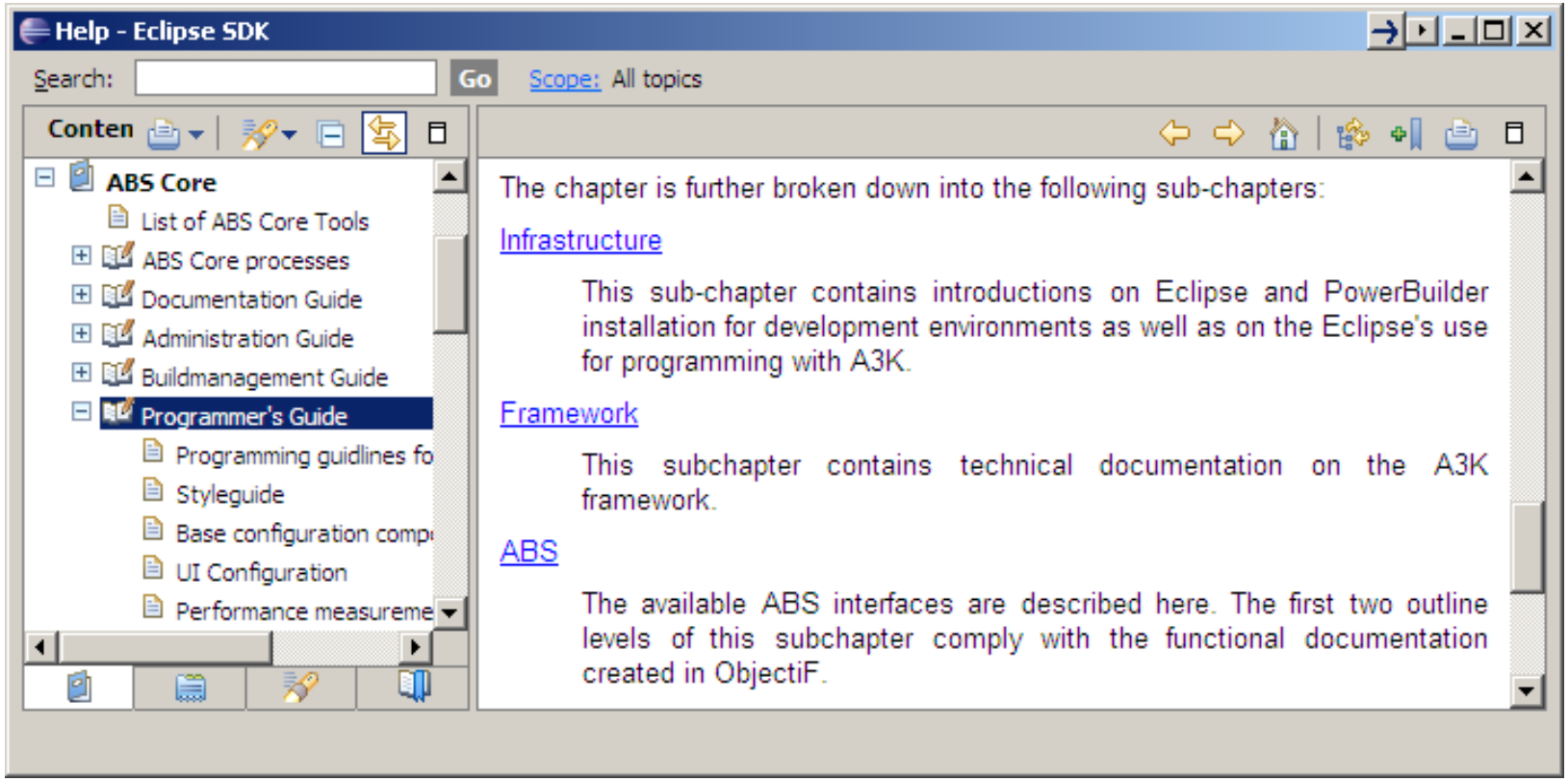


Allianz

1

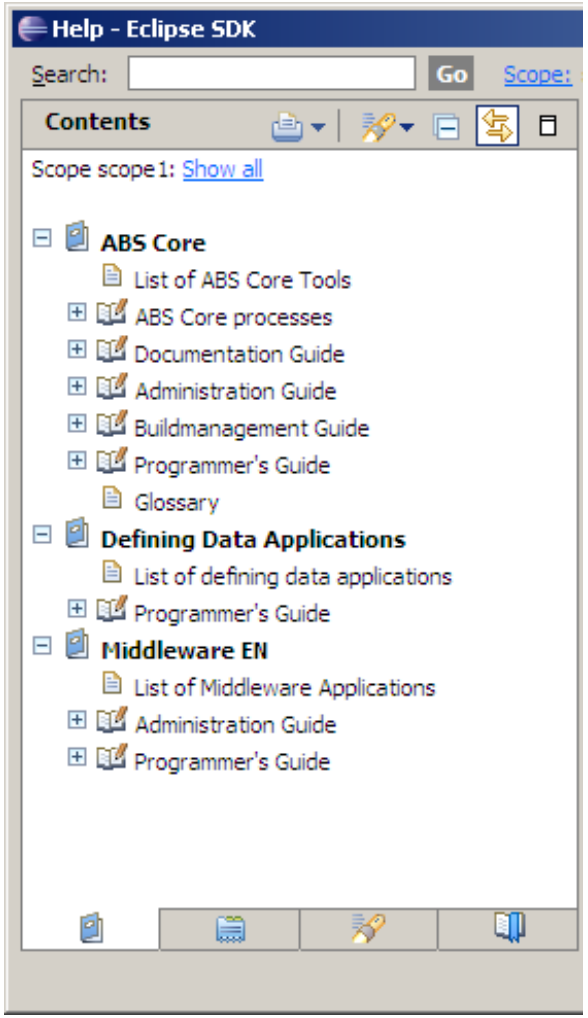
Documentation

- Installed in Eclipse Help as a plug-in
- Documentation plug-in at.allianz.a3k.doc.user



Documentation – structure

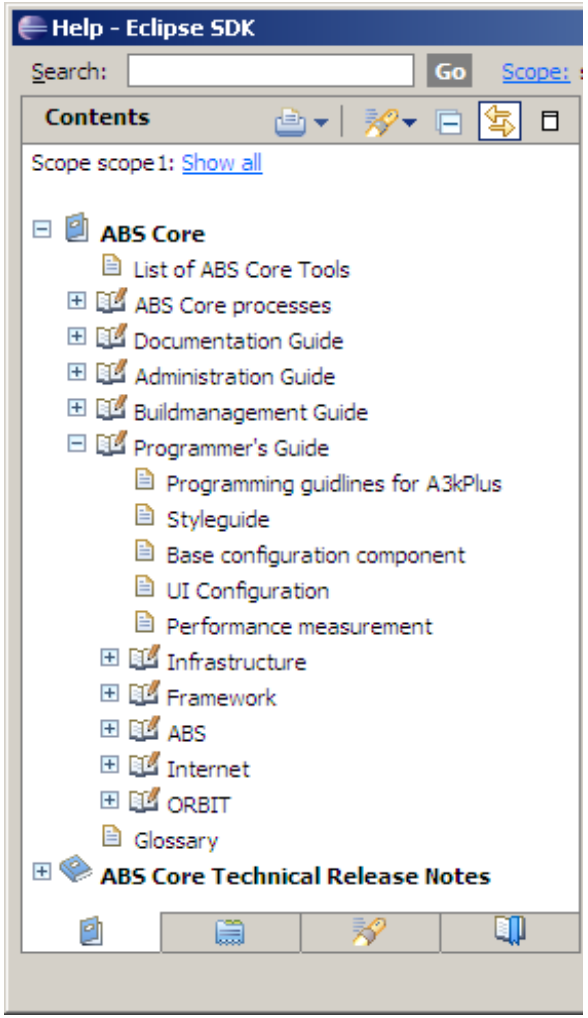
- ABS Core
  - → Programmer's Guide:
    All about A3k and ABS Java development
- Defining Data Applications
- Middleware
  - C++ components (“tanking”)



Allianz

Documentation – Programmer's guide

- Infrastructure
  - All about the development infrastructure
- Framework
  - Concepts and basic knowledge for development with A3k
- ABS
  - Technical documentation on functional areas
- Internet
- ORBIT



Allianz

Programming guidelines

- Core programming guidelines
  - Can be found under Programmer's guide
  - New code has to be developed in line with these guidelines
- Customer-specific programming guidelines

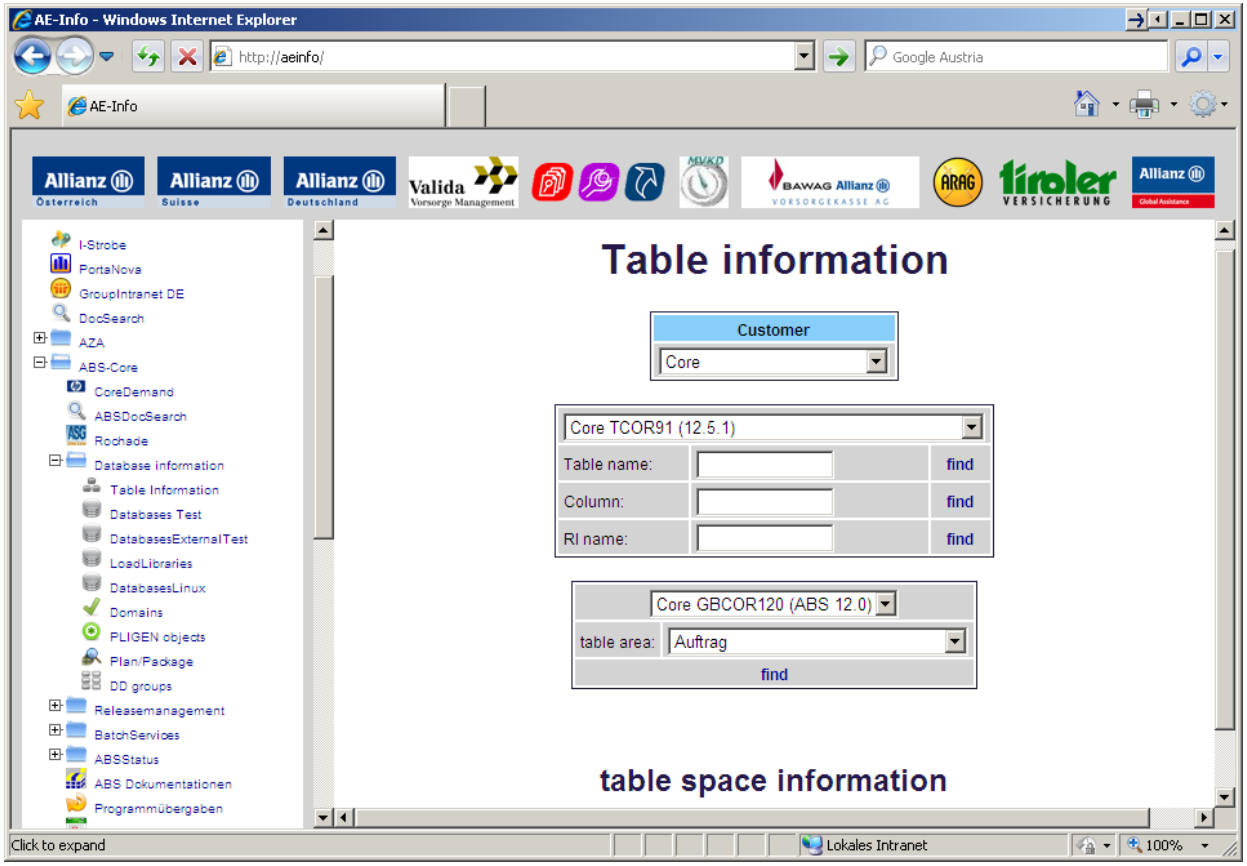
Allianz

Concepts, use cases and requirements

- Concepts
  - Document contains a list of concepts from A3k
  - Each concept is summarised on one page
  - Links to code references and dependent concepts
- Requirements and use cases
  - Document contains a list of typical requirements with links to concepts and use cases
  - Use cases are sample implementations of these requirements

ABS Info/AE Info

Data model information, programming documentation etc.:
http://as-absinfo01.aeat.allianz.at/ or http://aeinfo.allianz.at.awin/



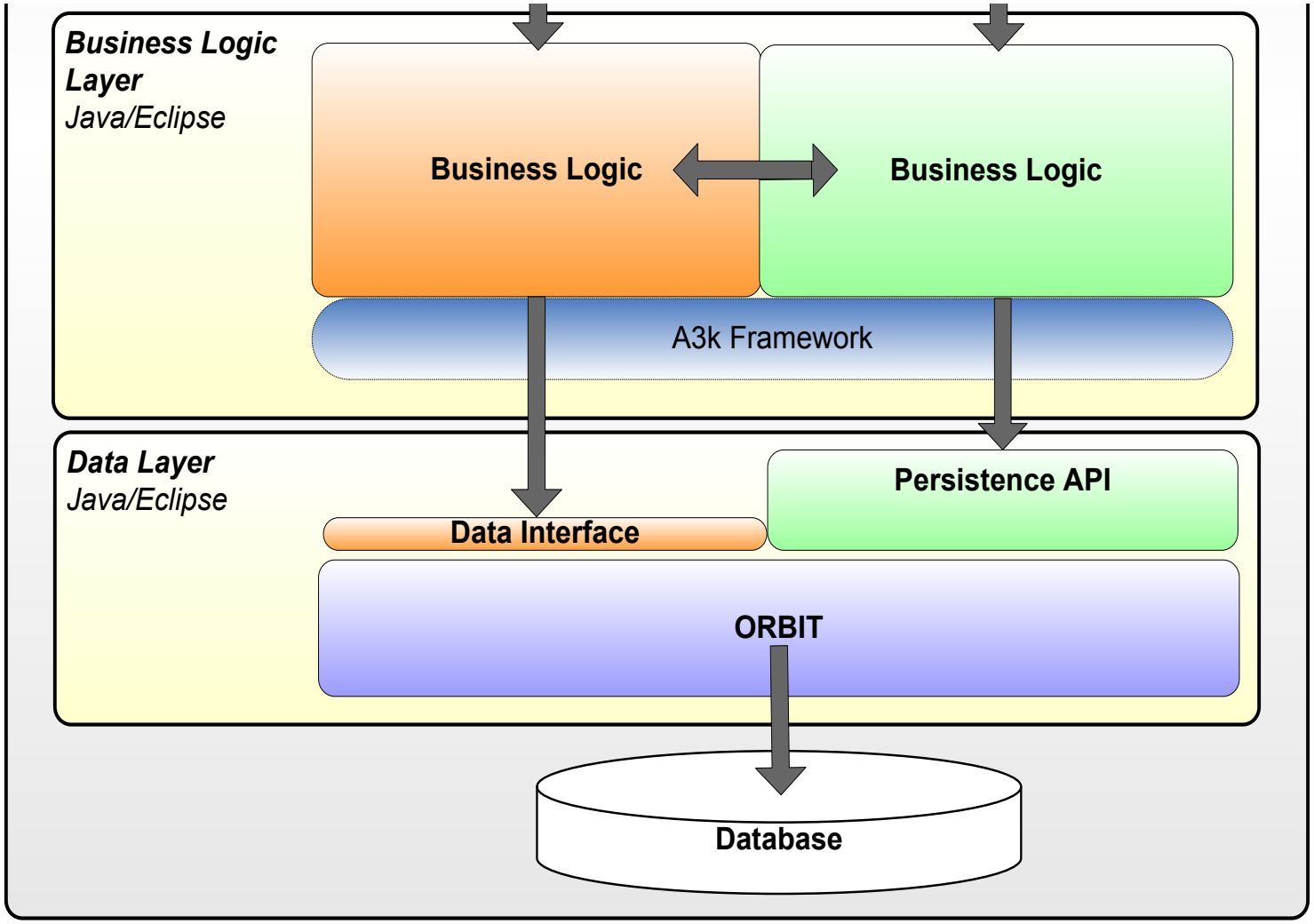
2

Allianz

2

Allianz

Overview of the architecture

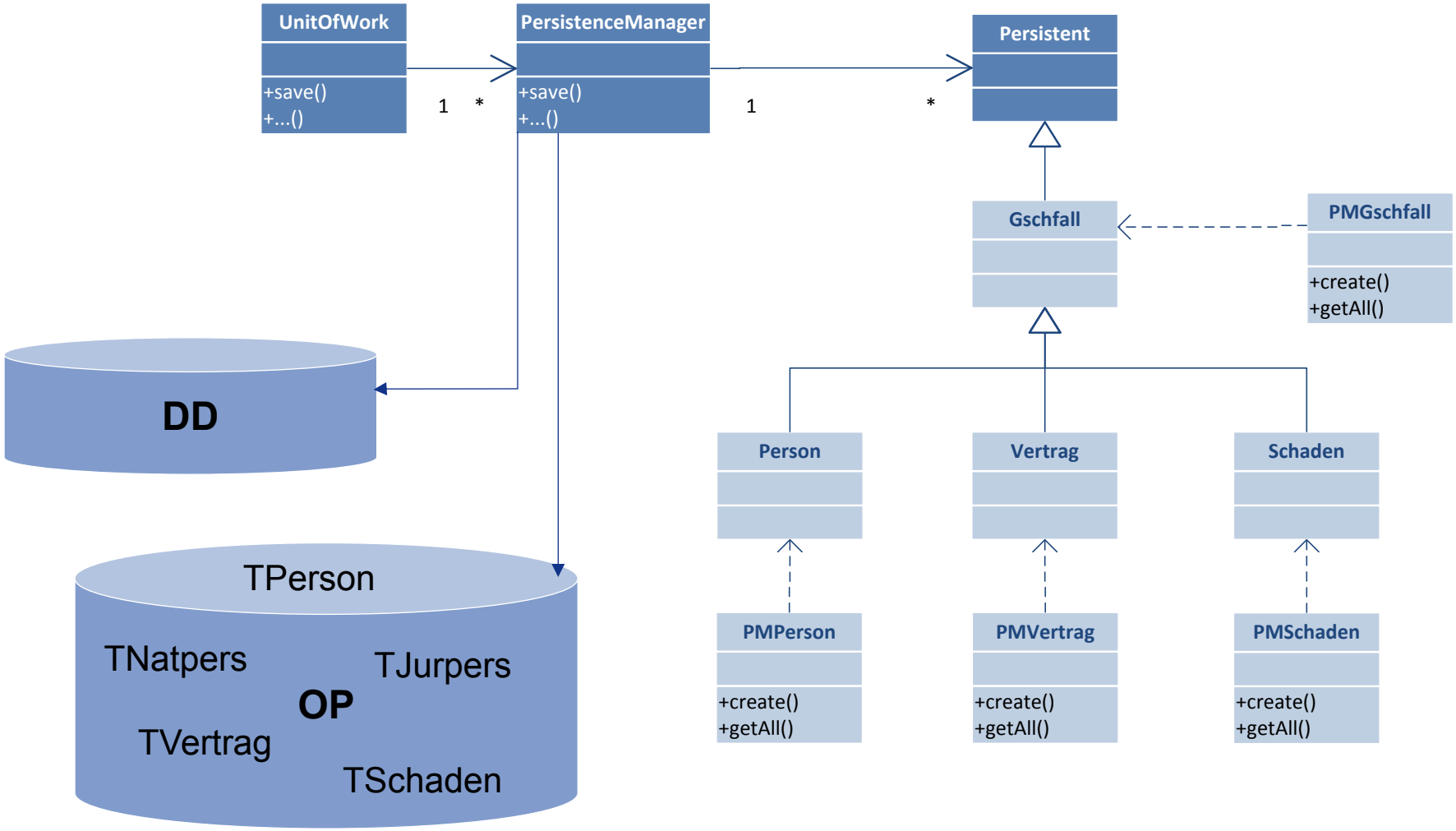


Allianz

Orbit

- Abstraction from the data source (databases, historisation, etc.)
- Object-oriented view of the business object data
- Strict separation of data management and functional logic: i.e. functional logic is not to be implemented in Orbit
- The data management is a clearly defined component which can be used by several applications, possibly on different computers, if necessary.
- Orbit manages objects in a strict hierarchy
  - There are “root” and “child” objects
  - A child object can’t exist without its root object, neither in the database nor in Orbit
  - If you directly load a child object, Orbit automatically loads its parent

Persistence layer



Allianz

ABS data model

- Data model structure
  - The data model has an object-oriented structure
  - Within the gfb_schema it is possible to design foreign key references as inheritance or aggregation:
    - Aggregation is mapped to relations and the aggregated class has a foreign key reference to its parent
    - A derived class always has a foreign key reference to its superclass

Allianz

ABS data model

Domains

- Domain = defined set of values
- Data model stores information showing the domain that a table attribute belongs to
- To the individual values in a domain definition belong:
  - Short text (value)
  - Long text
  - Printed long text (optional)
  - Validity period (valid from/to)
- Domain definitions are part of the defining data
- An entity attribute which belongs to a specific domain can only have a value which is defined in this domain.

Allianz

Persistence layer

PersistenceManager

- Manages a database connection
- ABS uses separate databases and therefore two (or more) Persistence Managers:
  - OP: operational data
    - E.g. contracts, persons, addresses
    - Current and historical data
  - GGRL: defining data
    - Configuration
    - Rights and authorizations
    - Domain definitions

Allianz

Persistence layer

A3kSession: user session

- Represents a user's space in the application
- Objects belonging to the user are linked to the A3kSession

UnitOfWork

- Represents a file window
- Keeps several PersistenceManagers
- "Save bracket": Knows the changes to be saved
- Contains the logical application area
- UnitOfWork mapping: context-specific mapping of adapters and persistent delegates

Allianz

Persistence layer

Generated interfaces and classes:

- Interfaces for persistent objects
  - Representation of a database table as a Java object (interface)
    - Example `Natpers:` represents `TNATPERS/HNATPERS`
- PM Persistence classes: static lifecycle methods and column definitions
  - Creation of new objects
  - Loading of existing objects from the database:
    - A single object with the primary key (`DbKey`)
    - A single object with a unique attribute
    - A list of objects with a filter
  - Navigation to other persistent objects
  - Access methods for domains
- Domain interfaces: constants with the domain values

Allianz

Persistence layer – persistent object interfaces

Generated for each table: interface and Orbit implementation

- Application code works solely with interfaces
- e.g. `Person` and `OrbitPerson`
  - Constants as additional information

```
public interface Person extends Gschfall {
String PCLASSNAME = "Person";
String TABLENAME = "TPERSON";
String TABLENAME_FULL = (FLQ.length() > 0 ? FLQ + "." : "")+TABLENAME;
```

  - Getters/setters for the object attributes

```
String KZPERS = "KZPERS";
String KZPERS_FULL = "WAG.TPERSON.KZPERS";
String getKzpers();
String getKzpersOriginal();
IDomain getKzpersDomain();
void setKzpers(String value);
```

Allianz

Persistence layer – persistent object interfaces

- Getters for related objects
  if the related object is in the class path; otherwise: see PM classes

```
public final ChildPersistentRelation<Adresse> getAdresse()
```

Allianz

DbKey

- The `DbKey` object represents the database key for an object in Java
- The PM classes can be used to load persistent objects with a `DbKey` object
- Acceptable formats are:
  - 36 characters – UUID (or 32 characters without hyphens)
  - 26 characters – time stamp
- Anything else results in an exception
- Code example:

```
DbKey contractKey = new DbKey("2ad81e98-6058-11dc-8625-030aae6e00a9");
```

Allianz

Persistence layer – PM classes

PM classes only contain static methods/attributes

- Loading a single object using its primary key

```
public static Person get(final UnitOfWork unitOfWork,
        final String dbum, final DbKey primaryKey) {…}
```

- Loading a single object using a unique index attribute

```
public static Flotte getByFlottennr(final IPersistentContext
        persistentContext, final String value)
```

Allianz

Persistence layer – PM classes

- Loading a list of objects using a filter

```
public static List<Person> getAll(final UnitOfWork unitOfWork,
        final String dbum, final String filter) {…}
```

- Filter conditions refer to attributes of the persistent class or its superclass and use the corresponding constants.

  Examples:

```
List<Land_def> landDefList = PMLand_def.getAll(
      getUnitOfWork(), DbConnectionData.DEFAULT_DBUM,
      Land_def.POSTLANDSCHL + "='" + postlandschl + "'");
```

```
List<Tier> tierList = PMTier.getAll(
      getUnitOfWork(), DbConnectionData.DEFAULT_DBUM,
      Tier.TIERNAME + "='" + name + "'“);
```

Allianz

Persistence layer – PM classes

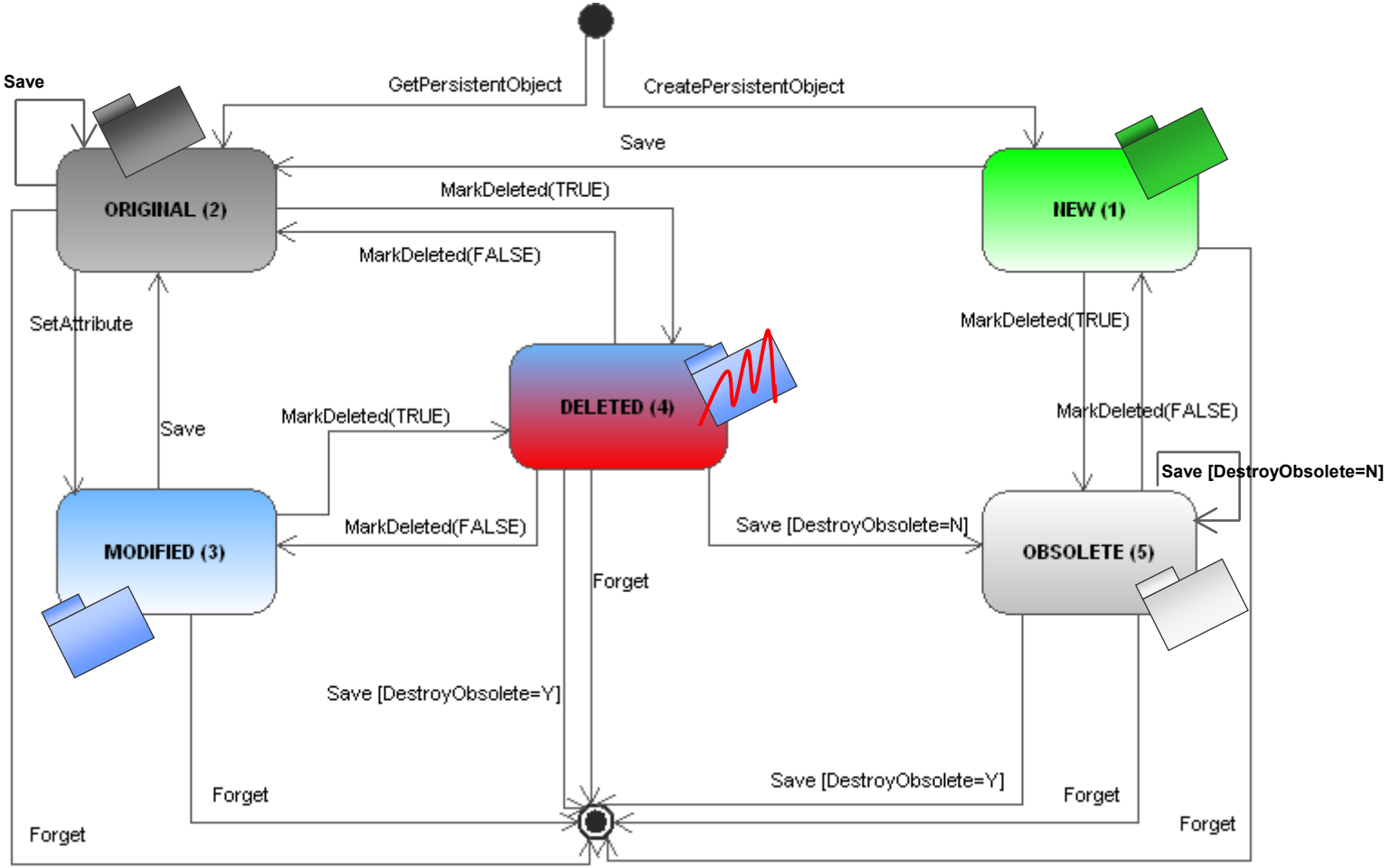
- Getters for `AttributeInfos`
  - Information which describes an attribute. e.g.:
    `AttributeInfo<String> PMNatpers.getFamnameInfo()`
  - Provide name, domain name, persistent class, table name, attribute type
- Navigation using relations
  (if the method cannot be in the persistent interface itself, due to plug-in borders)

```
public final static PersistentRelation<Person>
        getInkassoPersonForAdresse(final Adresse adresse,
        String filter, final boolean activeOnly) {…}
```

Allianz

The lifecycle of a persistent object

Possible object states and transitions



Allianz

The lifecycle of a persistent object

An example for a typical lifecycle

- Obtain an object
  - Load an existing object: `Tier tier = PMTier.get(…);`
  - Or create a new object: `Tier tier = PMTier.create(…);`
- Change some attributes:
  `tier.setTiername("Bello");`
- Maybe delete the object:
  `tier.markDeleted(true);`
- Save the object in the database (done by the core system, usually not explicitly by the application programmer):
  `unitOfWork.save();`

Quiz 1

Question 1

What are the layers of the ABS architecture?

a. Dialog layer
b. Business layer
c. Internet layer
d. Persistence layer
e. GUI layer
f. Functional layer
g. Portal layer

Question 2

There are several components important within the persistence layer. What is ORBIT?

a. Database underlying ABS
b. Data management tool for ABS and abstraction of different data sources
c. Layer where all functional logic is implemented
d. Used for navigation using relations

Allianz

Question 3

Remember the persistence layer and the different concepts used within. What is a domain?

a. Defined set of values
b. Printed long text
c. Database key for an object in java
d. Manages the connection to the database

Solution: a

Question 4

The Unit of Work (UOW) is the central management object within ABS. Select the different responsibilities of the UOW.

a. Represents a file window in ABS
b. Holds the session information
c. Manages database queries
d. Has a save method for saving persistent objects
e. Holds any number of persistence managers
f. Knows the sum of all persistent objects loaded and modified in the UOW

Question 5

What are characteristics of a persistent manager?

a. Responsible for the database connection
b. Generates the documentation of A3k
c. There are at least two persistence managers – one for the operational database and one for the defining database
d. New persistent objects are created via the persistence manager

Question 6

A persistent object can have different states. What are possible states?

a. Obsolete
b. Closed
c. New
d. Deleted
e. Selected
f. Modified
g. Valid
h. Archived

Solution: a, c, d, f

Allianz

Exercise 2.1.1: Preparations

Preparations for the exercises: Create a new plug-in with a test class

- A new plug-in project `at.allianz.gfb.core.training.persistence` should be created.
- A new test class called `PersistenceTest` should serve as a container for the executable code of the following exercises.

Note: JUnit is not being used for module test in this case, but rather as a simple way of making code executable.

Allianz

Exercise 2.1.2: Persistent objects and PM classes

Load persistent objects

- Some persistent objects of natural persons (Natpers) are to be loaded via PM classes (e.g. PMNatpers). The first name (‘Vorname’) and surname (‘Famname’) of each person is then to be displayed on the console.

Load the following natural persons:

- The person with the NP#
  58F760D2-853D-11D1-8000-010157AA0000
- The person with the PERS#
  4E8F096A-5D42-11CC-8000-010157AA0000
- All persons with the surname ‘Schmied’
- The person with the customer number (‘Kundennr’) 10001160

Allianz

ORBIT debugging: Voyager

- Is available as a separate standalone tool
- Serves the following purposes for debugging:
  Visualisation of data and structures for ORBIT, from any process
  (with the JMX debug option)
  - ‘Data’ screen: Shows currently existing objects in the Orbit cache with their data, with the option of editing data
  - ‘Views’ screen: SQL statements for views, sent to the DB during the runtime of Voyager
  - ‘Schema’ screen: Shows the classes/objects of the data model and their relations as an interactive diagram

Demo

Allianz

ORBIT debugging: Log files

ORBIT can be instructed to log the SQL sent to the database into files

- Setting a log file path switches the logging on (registry path on Windows 64 bit):
  ```
  [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Allianz\ORBIT\PATH]
  /SEL_PROT_FILE="C:\temp\logs\ORBIT\sel_prot.txt"
  /SQL_PROT_FILE="C:\temp\logs\ORBIT\sql_prot.txt"
  /DB2LOG="C:\temp\logs\ORBIT\db2log.txt"
  /GF_PROT_FILE="C:\temp\logs\ORBIT\gf_prot.txt"
  /HIST_PROT_FILE="C:\temp\logs\ORBIT\hist_prot.txt"
  /ZDB_PROT_FILE="C:\temp\logs\ORBIT\zdb_prot_file.txt"
  /VIEWDEF_PROT_FILE="C:\temp\logs\ORBIT\viewdef_prot.txt"
  /DB2_TIME_FILE="C:\temp\logs\ORBIT\db2_time.txt“
  ```
- Most interesting:
  `sel_prot.txt`: SELECT statements (direct database calls)
  `sql_prot.txt`: INSERT, UPDATE statements (via stored procedures)
  `db2log.txt`: Complete data sent to and received from the database
- To additionally record the stack:
  ```
  [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Allianz\ORBIT\SWITCHES]
  /SEL_PROT_STACKTRACE="ON"
  ```

Allianz

DataContainer – Features

- Tabular data management in several buffers
- Each column has a specified type
- Sorting, searching and filtering data
- Iteration over the data
- Columns calculated once
- Dynamically calculated columns
- Index and ID column
- Import of data from various sources such as collections, arrays, strings and other DataContainer columns
- Supports dynamic extensions

Allianz

DataContainer – When to use?

The A3k framework uses DataContainers for the following purposes:

- Persistence layer:
  - `DcDomain` for domain value sets
  - `View` for database queries
- Dialog layer:
  - `DoList` needs the underlying data of lists and tables in a DataContainer
- In other cases: it is recommended to prefer the Java Collections framework instead

Allianz

DataContainer – How to use?

- DataContainer successors represent the structural information
- Management of the column objects:
  - Column objects are produced only once at the time of creation (in the DataContainer constructor)
  - Column objects must be kept in instance variables
  - Getter methods make the structure available externally
- Columns have a specific type which can be any Java object
- Naming convention: DataContainer successor should begin with the prefix ‘Dc’
- No business logic in DataContainer successors!

Allianz

DataContainer – Definition of a DC

```
public class DcPerson extends DataContainer {

    private final Column<String> name;
    private final Column<DbDate> birthdate;

    public DcPerson(Locale locale) {
        super(locale);
        name = createColumn(String.class, "");
        birthdate= createColumn(DbDate.class);
    }

    public Column<String> getName() {
        return name;
    }

    public Column<DbDate> getBirthdate() {
        return birthdate;
    }

}
```

Allianz

DataContainer – Cell access

- Insert values

```
A3kSession session = ...;
DcPerson dc = new DcPerson(session.getLocale());

// Create new line and fill it with content
Row row = dc.createRow();
dc.getName().setValue(row, "Maier");
```

- Read values

```
// Query cell values
String name = dc.getName().getValue(row); // name = "Maier"
```

Allianz

DataContainer – Iterating and sorting

- Iterate over the content of a DC

```
DcPerson dc = new DcPerson(session.getLocale());

for (Row row : dc) {
    String name = dc.getName().getValue(row);
    DbDate gebdat = dc.getGebdat().getValue(row);
    // …
}
```

- Sorting

```
dc.getGebdat().setSortAscending(false);
dc.sort(dc.getName(), dc.getGebdat());
```

Sorting order is defined by the `Locale` of the DC

Allianz

Exercise 2.1.3: DataContainer

- Define a DataContainer
- A DataContainer should be defined and implemented.
- The DataContainer is then to be filled and read out.
  - A `DcPerson` class with columns for surname, first name and HR number should be created.
  - `DcPerson` should be filled with some rows with data.
  - Display the content of the DataContainer on the console.

Allianz

DataContainer – Filters

- Filtering data

```
DcPerson dcPerson = …

dcPerson.filter(new Filter() {

      @Override
      public boolean includeRow(Row row) {
            int nrOfChildren = dcPerson.getNrOfChildren().getValue(row);
            return nrOfChildren > 0;
      }
});
```

Package `at.allianz.a3k.datacontainer.filter` contains some predefined filter classes

Allianz

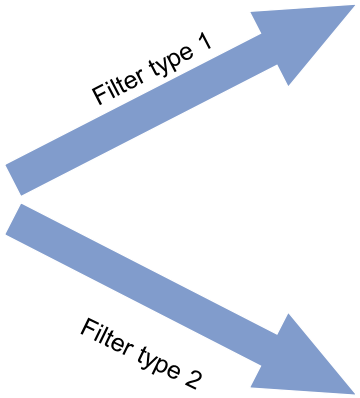
DataContainer – RowBuffers

RowBuffer =

a list of rows in the DataContainer

C **RowBuffer 1** D

Name	Children
Weber	1
Bauer	0
Maier	3
Huber	1



Filter: Children >0

keepExcludedData = false

C **RowBuffer 1** D

Name	Children
Weber	1
Maier	3
Huber	1

Filter: Children >0

keepExcludedData = true

C **RowBuffer 2**

Name	Children
Weber	1
Maier	3
Huber	1

D **RowBuffer 1**

Name	Children
Bauer	0

C Current RowBuffer

D Default RowBuffer

Rollback with the help of mergeAllRowBuffers()

```
dc.filter(true, hasChildrenFilter);
```

Allianz

DataContainer – Searching

- Searching for data

```
DcPerson dc = …;
FindResult result = dc.find(new
        GreaterZeroFilter(dc.getNrOfChildren()));
for (Row row : result) {
    // …
}
```

Note that `findFirst` might be faster than `find` and `find` might be faster than `filter`!

Allianz

DataContainer – Import

- Importing tab-separated string data

```
DcPerson dc = new DcPerson(session.getLocale());
dc.importString("Bauer\t01.01.2000\r\n"+
                "Maier\t24.12.1990",
                dc.getName(),
                dc.getBirthdate());
```

Note: This generally makes only sense for A2k data, not for new code!

Allianz

DataContainer – DcDomain

- Domain values are kept in a special type of DataContainer, `DcDomain`:
  - PM classes contain generated static methods:

```
public static IDomain getGeschlechtDomain(final A3kSession session) {
    return DomainProviderFactory.get(session)
            .getDomain(Natpers.GESCHLECHT_FULL);
}
```

  - Persistent object interfaces contain generated instance methods:

```
IDomain getGeschlechtDomain();
```

  - Provide the valid values of the domain “GESCHLECHT” (gender):

```
public DcDomain getGeschlechtDomain() {
    return PMNatpers.getGeschlechtDomain(getSession()).provideValid();
}
```

  - Default value can be fetched with `getVorbelegung()`

Allianz

Exercise 2.1.4: DcDomain

- Use domains
  - In this exercise, you have to read a domain from the database in the form of a DcDomain object.
    - Read the domain ‘Gesellschform’ (legal form) and display all valid values and display values on the console.

Allianz

Views

Dynamic view = database query in Java code

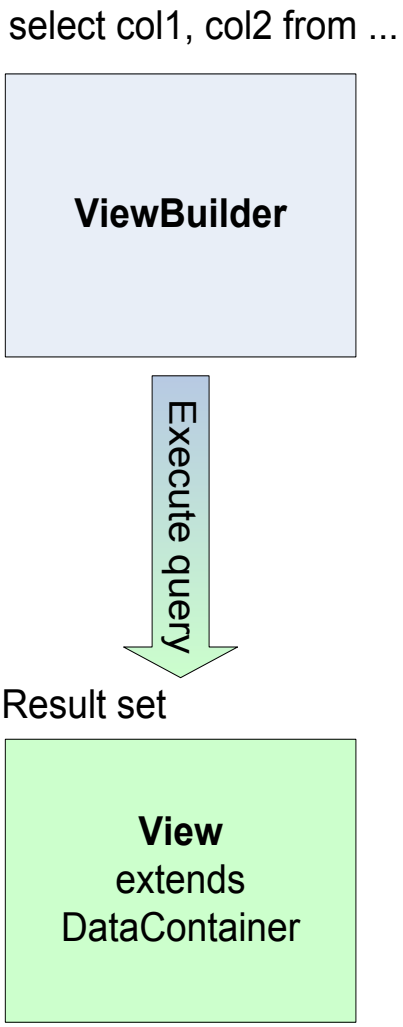
ViewBuilder class

- Describe DB query
  - Result columns
  - Selection criteria (conditions)
  - Joins (have to be explicitly specified)
- Execute DB query

View class

- Container for query results (DataContainer)
- Own class with column definition can be derived from this

Note: This is not related with the ‘view’ concept of the same name in relational databases!



Allianz

Views – a simple example

```
// Define query using ViewBuilder
ViewBuilder viewBuilder = new ViewBuilder();
viewBuilder.addResultElement(PMNatpers.getFamnameInfo(),
        PMNatpers.getVornameInfo(), PMAdresse.getPostleitzahlInfo());

viewBuilder.join(PMNatpers.getPersKeyInfo(), PMAdresse.getPersKeyInfo());
viewBuilder.setCondition(PMNatpers.getSurnameInfo() + "= 'Maier' AND "
        + PMNatpers.getVornameInfo() + " LIKE 'Ma%'");
viewBuilder.setOrderBy(PMNatpers.getFamnameInfo(), true,
        PMNatpers.getVornameInfo(), true);

// Run query
View view = new View(unitOfWork.getPersistenceManager(DbConnectionData.DEFAULT_DBUM,
        "OP"));
viewBuilder.performView(view);

// Access to data
for (Row row : view) {
    String surname = view.getColumn(PMNatpers.getFamnameInfo()).getValue(row);
    String firstname = view.getColumn(PMNatpers.getVornameInfo()).getValue(row);
    String postalCode =
            view.getColumn(PMAdresse.getPostleitzahlInfo()).getValue(row);
    // ...
}
```

Allianz

Views – more complex queries

Support for more complex database queries

- Functions and constants
  - Classes `QueryFunction, QueryCount, QueryConstant` etc.
  - Can be used as a result element:

```
QueryCount count = new QueryCount(Tier.class);
viewBuilder.addResultElement(count);
// …
Row row = result.getRow(0);
int number = result.getColumn(count).getValue(row);
```

- Inner, Left Outer, Right Outer and Full Outer Joins
- Group By and having clause
- SQL Union, Except and Intersect
- Because of the DB design it is not possible to mix GGRL and OP data in a join (it will cause a runtime exception!)

Allianz

Views – joins

Creating conditions with the Cond class

- The class Cond makes it easier to formulate a string based condition which is free of syntax errors.
- Therefore it is a good idea to use the class Cond.

```
List<DbKey> personenGFKeys = …;
String filter = Cond.in(PMPerson.getGfKeyInfo(),
personenGFKeys);
viewBuilder.setCondition(filter);
// Alternative: loop for string concatenation
```

- If you need a logical operator like in, not, and, or, .. you can to use the Cond class as an alternative to a string concatenation.

Allianz

Views – own view classes

Derivation of own view classes

- Similar procedure to that for DataContainer:
  - Own class derived from view
  - Get methods for column objects
  - The best practice is to initialise the column objects in the constructor
- Makes sense if the query result subsequently is required in table form

Allianz

Views – Example with own view class

```
public class ViewPerson extends View {

    private final Column<String> surname;
    private final Column<String> firstname;
    private final Column<String> postalcode;

    public ViewPerson(PersistenceManager pm) {
        super(pm);
        surname = createViewColumn(PMNatpers.getFamnameInfo());
        firstname = createViewColumn(PMNatpers.getVornameInfo());
        postalcode = createViewColumn(PMAdresse.getPostleitzahlInfo());
    }

    public Column<String> getSurname() {
        return surname;
    }

    public Column<String> getFirstname() {
        return firstname;
    }

    public Column<String> getPostalCode() {
        return postalcode;
    }
}
```

Allianz

Views – Example with own view class

```
// Define query using ViewBuilder
ViewBuilder viewBuilder = new ViewBuilder();
viewBuilder.addResultElement(PMNatpers.getFamnameInfo(),
        PMNatpers.getVornameInfo(), PMAdresse.getPostleitzahlInfo());

viewBuilder.join(PMNatpers.getPersKeyInfo(), PMAdresse.getPersKeyInfo());
viewBuilder.setCondition(PMNatpers.getSurnameInfo() + "= 'Maier' AND "
        + PMNatpers.getVornameInfo() + " LIKE 'Ma%'");
viewBuilder.setOrderBy(PMNatpers.getFamnameInfo(), true,
        PMNatpers.getVornameInfo(), true);

// Run query, result in own view class
ViewPerson viewPerson = new ViewPerson(
        unitOfWork.getPersistenceManager(DbConnectionData.DEFAULT_DBUM, "OP"));
viewBuilder.performView(viewPerson);

// Access to the data via own view class
for (Row row : viewPerson) {
    String surname = viewPerson.getSurname().getValue(row);
    String firstname = viewPerson.getFirstname().getValue(row);
    String postalcode = viewPerson.getPostalcode().getValue(row);
    // ...
}
```

Allianz

Comparison of views and PM classes

ViewBuilder/View	PM classes/persistent objects
▪ Returns only explicitly requested attributes	▪ Produces complete objects with all attributes, also from super classes
▪ Can only be used for pure data reading, has no storage option	▪ Delivered objects can be changed using set methods and can be saved via `UnitofWork`
▪ Query goes straight to the database with no cache consideration	▪ Takes the Orbit cache into account, including unsaved changes
▪ Complex conditions possible	▪ Only limited filter options
▪ Should not be used for navigation between objects	▪ Contains methods for navigation between related objects

Quiz 2

Question 1

What is the purpose of the tool Voyager?

a. Debugging ORBIT
b. Transferring data via the web
c. Creating reports on system load
d. Visualisation of data and structures

Solution:

Solution: a, d

Question 2

Do you remember the features of a data container?

a. Component for GUI design
b. Data management component in tabular form
c. Sorting, searching and filtering methods are provided
d. Backup of the ABS database
e. Dynamically calculated columns
f. Index and ID column
g. Manages session data
h. Extendable
i. Iterable

Solution: b, c, e, f, h, i

Question 3

In A3k we are using data containers only in certain cases. What are proper use cases?

a. Domains
b. Database queries in form of views
c. Creating new persistent objects
d. DoList in the dialog layer
e. Data archiving

Solution: a, b, d

Question 4

The content of a data container can be filtered. In which method of the 'FILTER' class the filter criteria is defined?

a. equalsZero
b. isFiltered
c. hasChild
d. includeRow

Solution: d

Question 5

The data container works with row buffers. What characteristics does a row buffer have?

a. It is always empty
b. A data container has at least one row buffer
c. Holds a number of rows of a data container
d. Works as cache for ORBIT
e. Any numbers of row buffers can be created via createRowBuffer()

Solution: b, c, e

Question 6

Domain values are kept in a special type of data container. How is it called?

a. DcDomain
b. DomainContainer
c. DataDomain
d. ValueDomainSet

Solution: a

Allianz

Question 7

A further use case of data containers are views. Views are database queries in Java code. Which two classes are important when working with views?

a. QueryBuilder
b. ViewCreator
c. SQLViewer
d. ViewBuilder
e. ResultContainer
f. View

Solution: d, f

Question 8

Remember the comparison of views and persistence manager classes? Which statements are correct?

a. Views return complete objects; persistence manager classes return only explicitly requested attributes
b. Views can only be used for data reading operations and have no storage option
c. Persistence manager classes take the Orbit cache into account
d. The navigation between objects can only be done via views

Solution: b, c

Allianz

Exercise 2.1.5: View query

- Executing view queries
  - A view should be executed on the ABS database using a `ViewBuilder`.
    - The company names (Firmenname) of all legal persons (`Jurpers`) are to be determined.
    - These are then to be displayed on the console.

Allianz

Exercise 2.1.6: Sorting and filtering in view queries

- View queries with filter and sort
  - Run a view on the database using ViewBuilder that returns the first names (‘Vorname’) and surnames (‘Famname’) of some natural persons (Natpers). Display the result sets on the console.
    - First step: Only natural persons with the surname ‘Krammer’ are to be selected. The data in the view should be sorted by first name in ascending order.
    - In a second step, extend the restriction to include only persons whose first name starts with an ‘R’..

Allianz

Exercise 2.1.7: Sorting and filtering DataContainer content

- Sort and filter a DataContainer
- Run a view on the database that returns the first and surnames of all
- Natural persons.
  - The DataContainer filter and sort functions are to be used to produce the same results as in the previous exercise. This means that filters have to be implemented and applied to the DataContainer. The results then need to be sorted accordingly.
  - The second step should implement a second filter that further restricts the results to include only first names starting with ‘R’ and then to apply this to the view as well.

The results must match the output of the previous exercise.

Allianz

Exercise 2.1.8: Functions and joins

- Use QueryFunctions and Joins
  - First of all, find out the number of all natural persons in the database whose surname starts with A.
  - Then use another, separate database query to find out the surname (Familienname), first name (Vorname), account manager number (Betreuer) and postcode (Postleitzahl) of these individuals. Limit the number of results to a maximum of 50 hits.

2

Allianz

Business logic layer (business layer)

- Plug-in component structure
- Eclipse adapter concept
- Business objects and unmanaged objects
- UnitOfWork
- Inheritance in the business layer
- API vs. internal
- Business attributes (BoAttributes)
- Validation and access restrictions
- Integration with A2k and integration adapters

Allianz

The Eclipse plug-in concept

“Everything is an extension”
(Erich Gamma, Kent Beck, “Contributing to Eclipse”)

- Eclipse and ABS aren’t monolithic systems, but are composed of many modules
- These modules are called *plug-ins* (Eclipse term) or *bundles* (OSGi term)

Allianz

The Eclipse plug-in concept: Key features

Characteristics:

- Modularisation
  - Modules form the structure of the system
  - Customers can decide what modules to install
- Defined API
  - Published code: Exported Java packages, visible to the outside
  - Internal parts: Packages which are not exported
- Defined dependencies
  - Each plug-in defines its dependencies
  - No explicit class path configuration

Allianz

Plug-in component structure

- ABS application is divided into several plug-ins, by functional aspects:
  - Framework, base
  - Person, contract, claim, feedback, journal etc.
- Functional modules consist of
  - Business logic,
    e.g. `at.allianz.gfb.core.person`
  - Dialog and GUI logic,
    e.g. `at.allianz.gfb.core.person.gui`
  - JUnit tests for the business logic (during development),
    e.g. `at.allianz.gfb.core.person.test`

Allianz

Conventions for plug-ins in A3k

- Singleton option should be set
- Activator class should contain a constant PLUGIN_ID
- Dependencies:
  - Use plug-in dependencies, not package imports
  - Always set the reexport flag
- Do not use version specifications
- Plug-in name = plug-in ID = root package name

Note: Exceptions to these recommendations are okay, if they have a reason.

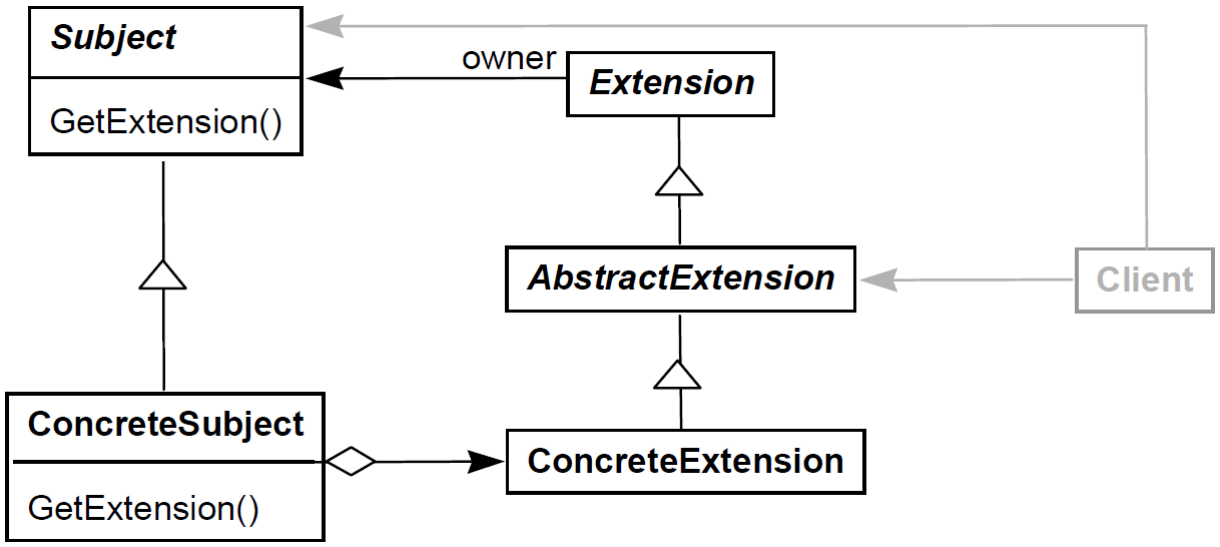
Allianz

The Extension Objects pattern

Design pattern, published by Erich Gamma (1996)

Source: http://www.mif.vu.lt/~plukas/resources/Extension%20Objects/ExtensionObjectsPattern%20Gamma96.pdf

“Anticipate that an object’s interface needs to be extended in the future. Additional interfaces are defined by extension objects.”



Allianz

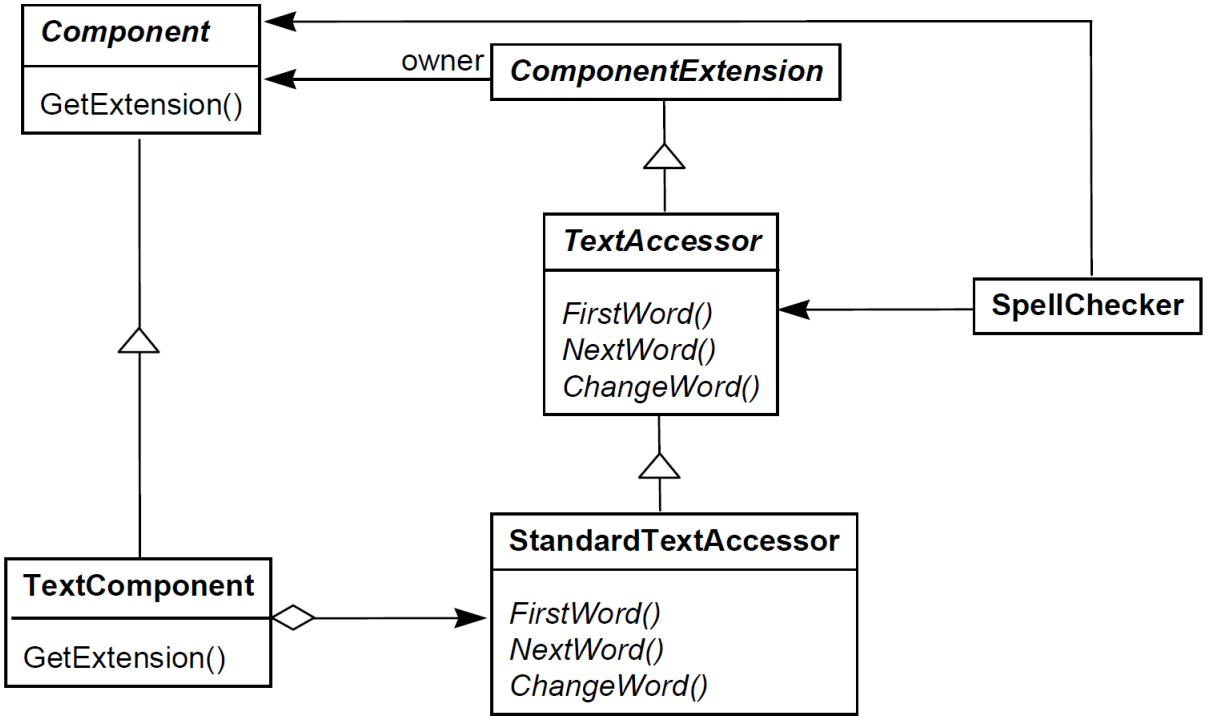
The Extension Objects pattern

- Assembly of classes with a common interface, with the base class '`Subject`'
- It should be possible to add interfaces which define additional, unforeseen functionality separately, without having to extend Subject
- The common interface for such extensions is called '`Extension`'. An extension knows its owning subject
- Additional functionality can be defined in an interface '`AbstractExtension`'
- A client who wants to use such additional functionality can query whether a subject supports it:
  It calls '`getExtension`' with the extension interface name
- If the subject provides this functionality, it returns an extension object which implements this interface. If not, it returns 'null'.

Allianz

The Extension Objects pattern: An example

- A compound document architecture, like OLE 2 or OpenDoc, with documents made of components like text, graphics, spreadsheets, movies
- Spelling checker for a compound document needs to enumerate the words of components that store texts



Allianz

The Extension Objects pattern: Applicability, implementation issues

Use the Extension Objects pattern when:

- The addition of new or unforeseen interfaces to existing classes should be supported, without impact on clients which don't need these
- A class should be extensible with new behaviour without subclassing from it
- A class representing a key abstraction plays different roles for different clients

Implementation issues:

- Extensions need to be created and managed
- Subject should not need to know its extensions forehand → subject maintains a dictionary of extensions; new extensions can be added on demand

Allianz

Eclipse Adapter pattern = Extension Objects pattern

The Eclipse platform is designed for extensibility and uses this pattern heavily

- Different terminology, but same pattern:
  - A class designed for extensibility:
    'Subject' → 'Adaptable'
  - Additional functionality:
    'Extension' → 'Adapter'
  - Fetch an extension:
    '`getExtension`' → '`getAdapter`'
  - Wording for adapter relationships:
    'X is an extension for Y' → 'X adapts Y'
- Available Eclipse framework functionality:
  - Interface '`IAdaptable`' for all adaptable classes
  - '`IAdapterFactory`' for Adapter creation
  - '`AdapterManager`' maintains dictionary of adapters

Allianz

Application of the Adapter pattern in A3k

- ABS specialities:
  - Lifecycle:
    - Creation of an adapter just once, adapter keeps its state
    - Adapter object dies with its adaptable
  - Dictionary: Called ‘mapping’ or ‘registration of adapters’, once per `UnitOfWork` class
- Simplifications:
  - `getTypedAdapter`
    returns the adapter object in a typed fashion
    (`getAdapter` returns just ‘Object’ which needs to be typecasted)
  - `getCheckedAdapter`
    additionally performs a null check (and throws an exception, if necessary)
    → recommended way to fetch an adapter
  - `BoReflectionAdapterFactory`
    is a generic adapter factory

Allianz

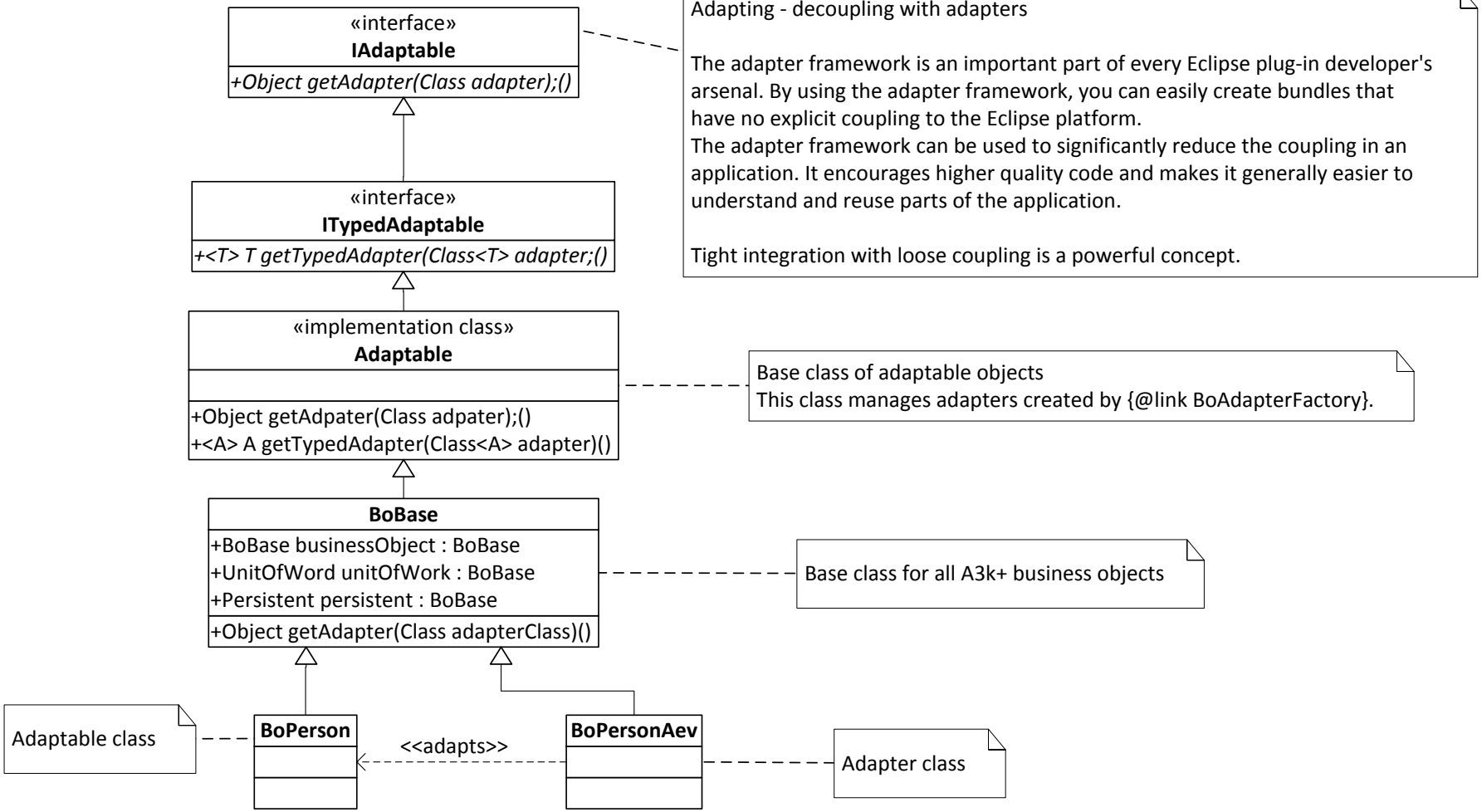
Use cases of the Adapter pattern in A3k

Different use cases:

- Business Adapter (or 'normal Adapter')
  - Most business objects are designed as adapters
  - In most cases business adapters just have an implementation class and do without an extension interface
- Implementation Adapter/Integration Adapter
  - Consists of a published interface (API) and a separated internal implementation class
  - For integration purposes
- Extension Adapter
  - Core system contains an extension interface and a default implementation
  - Customers can bring in their own implementation of the interface, Core calls customer-specific logic

Allianz

Eclipse adapters usage




```
BoPerson boPerson = ...;
BoPersonAev personAev = boPerson.getTypedAdapter(BoPersonAev.class);
```

Allianz

Business objects

- Get derived from `BoBase`
  - Know their `UnitOfWork` and the A3kSession
  - Know the persistent and/or business object to which they belong
- Class names have the prefix `Bo`
  - Examples: `BoPerson, BoNatpers, BoVertrag`

Allianz

Business objects as adapters

Design considerations:

- What should be adapted is determined by the lifecycle and the functional associations:
  - **Persistent object**: Determined by the functional association e.g. `BoAdresse` adapts the persistent object `Adresse`
  - **Business object**: Determined by the functional association or if the order of destruction is relevant
  - **UnitOfWork**: If the adapter should have the lifespan of the file window or the lifespan of a certain workflow in the web portals
- Constructor with object to be adapted (adaptable) as a parameter

Allianz

Implementation of a business logic class

- The business logic class requires a constructor with the object that has to be adapted as a parameter. This is passed on to a super constructor.
- `BoBase` remembers this and offers corresponding access methods depending on the type

Persistent	`getPersistent()`
BoBase	`getBusinessObject()`
UnitOfWork	`getUnitOfWork()`

- The relevant method should be overwritten with the specific adaptable type as return type.
- No need to additionally store the adapted object in the specific class

Allianz

Example of a business logic class

```
public class BoContractExample extends BoBase {
    /**
     * Constructor
     *
     * @param vertrag Contract object to be adapted
     */
    public BoContractExample(Vertrag contract) {
        super(contract);
    }


    /**
     * @return Adapted policy
     *
     * @see at.allianz.a3k.bo.BoBase#getPersistent()
     */
    @Override
    public Vertrag getPersistent() {
        return (Vertrag) super.getPersistent();
    }


    // Various business logic methods
    // ...
}
```

Allianz

Adapter lifecycle

Lifecycle is managed by the framework

- Object creation
  - Creation of adapter object when accessed for first time with `getTypedAdapter()` or `getCheckedAdapter()`
  - In the event of further accesses, the existing object gets returned
- Clean-up
  - To clean up, the framework calls `release` if its adaptable is released
  - `releaseSelf()` can be used as a `release` hook, if it is necessary to incorporate clean-up logic
  - Order of `releaseSelf()`: from adapter to adaptable
  - Note: Generally nothing has to be done, because the garbage collector cleans up itself. Only if you allocate native resources out of the garbage collectors' scope, you are responsible for clean-up.

Allianz

Objects which are not managed by the framework

Explicitly instantiated business objects

- Not every class that contains business logic needs to extend `BoBase`
- Means that they have to be released themselves again using a `release()` call
- Should be released from the object in which they were created

Unmanaged objects

- Not managed by the framework
- “Normal” Java objects
- Typical examples:
  - List elements of large lists
  - Use of class libraries

Allianz

UnitOfWork – range of functions

- Acts as a bracket for the user's editing action
  - Business logic counterpart of the file window
  - Can be in view mode or in edit mode
  - Keeps `PersistenceManagers` (separated for operative and defining data)
  - Knows which persistent objects were changed and are to be saved
  - Keeps the `ValidationManager` (for validation messages)
- Contains the logical application area ("logischer Anwendungsbereich") as a specialisation parameter, e.g. `LogAwbNatpers`, `LogAwbJurpers`
- Is derived in a file type-specific manner (also allowed for customers)
- Serves as a context for registering adapters and persistent delegates
- `UnitOfWorkDelegates` can get attached for lifecycle activities: `onOpen()`, `onSave()`, etc.

Allianz

UnitOfWork mapping

- Adapter registrations in A3k are not made globally, but for the validity area of a certain UnitOfWork
- **mapCoreAdapter(...)**
  - Exclusively for the core mapping from core object to core object
  - Mappings cannot be overridden
  - Not accessible for customers
- **mapStrategy(...)**
  - For core mappings which are meant to get overridden by the customer
- **mapLocalAdapter(...)**
  - For customer-specific mappings
  - Mapping of customer object to core object or customer object to customer object

Allianz

UnitOfWork mapping

- **mapLocalAdapter(...) specifically:**

```
public class MyUowMapper implements IUnitOfWorkMapper {

    @Override
    public void map(UnitOfWork unitOfWork) {

        // Example with BoReflectionAdapterFactory (common case)
        unitOfWork.mapLocalAdapter(Vertrag.class,
                new BoReflectionAdapterFactory(BoVertragExample.class));

    }
}
```

- **Extension Point “unitOfWorkMapper”:**
  - Parameter “unitOfWorkClass”: the UnitOfWork in which the mappings should take effect
  - Parameter “mapperClass”: name of the mapper class

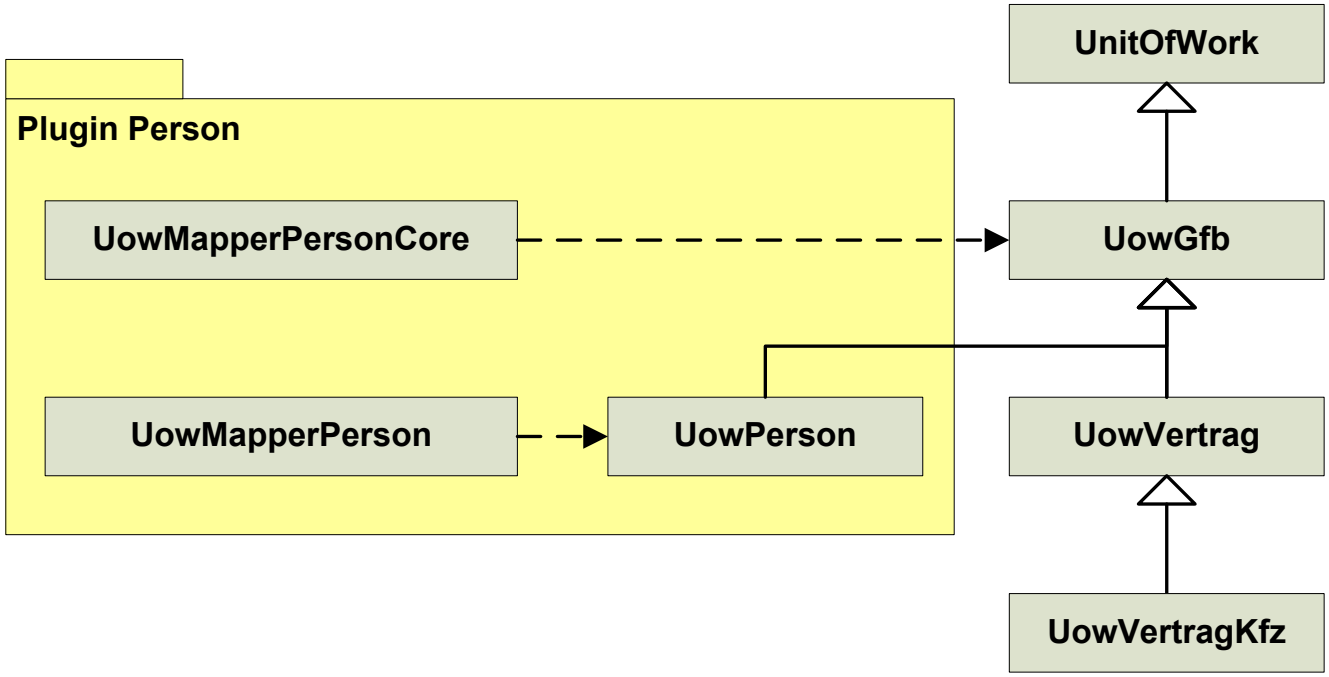
Allianz

UowGfb

- Abstract `Unitofwork` predecessor for all specific ABS files
- Defines basic mappings for all ABS key areas (person, contract, claim, ...)
- More than one mapper per area
  - Core mapper for general mappings
  - Special mapper for own area
  - Additional mappers are allowed

Allianz

UnitOfWork mappings with the UowGfb



- UowVertrag and UowVertragKfz see the general mapping from the person, e.g. Person → BoPerson

Allianz

Inheritance in the business layer

- Inheritance is only permitted based on functional aspects

  ✓ `BoNatpers extends BoPerson`

- Under no circumstances specific customer characteristics or special output media features (web, fat client ...) should be used in inheritance!

  DON'T `BoNatpersDe extends BoNatpers`

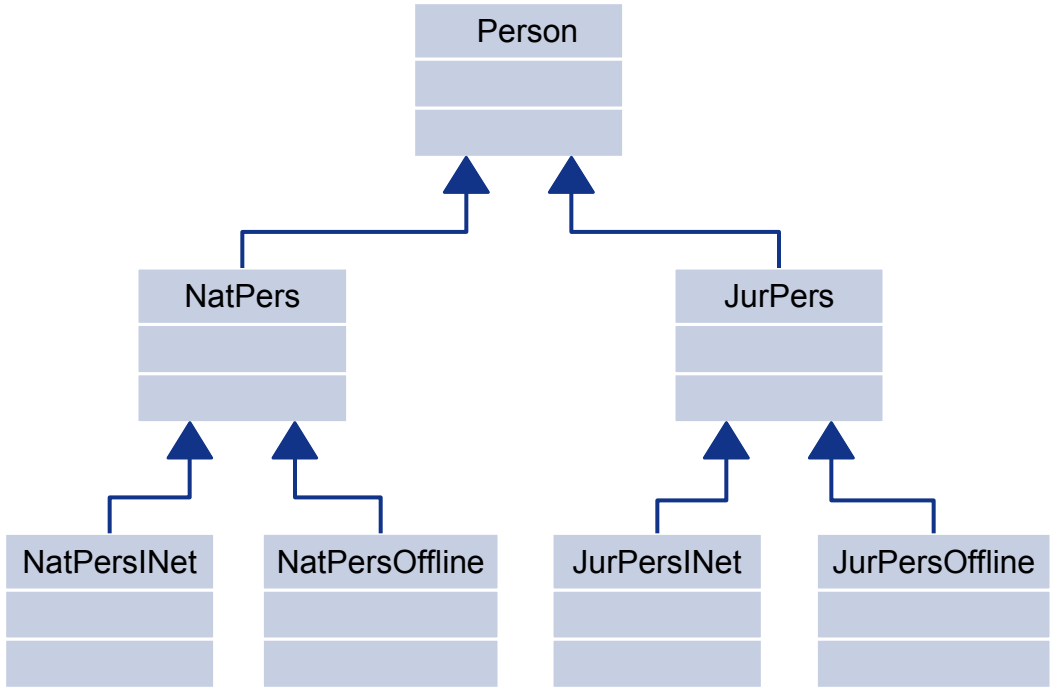
  → In such cases, adapt, don't derive!

  → No sub-class, but separate class with reference to the original

Allianz

Adaption vs. inheritance: a bad example

- 1st aspect
- 2nd aspect
- …

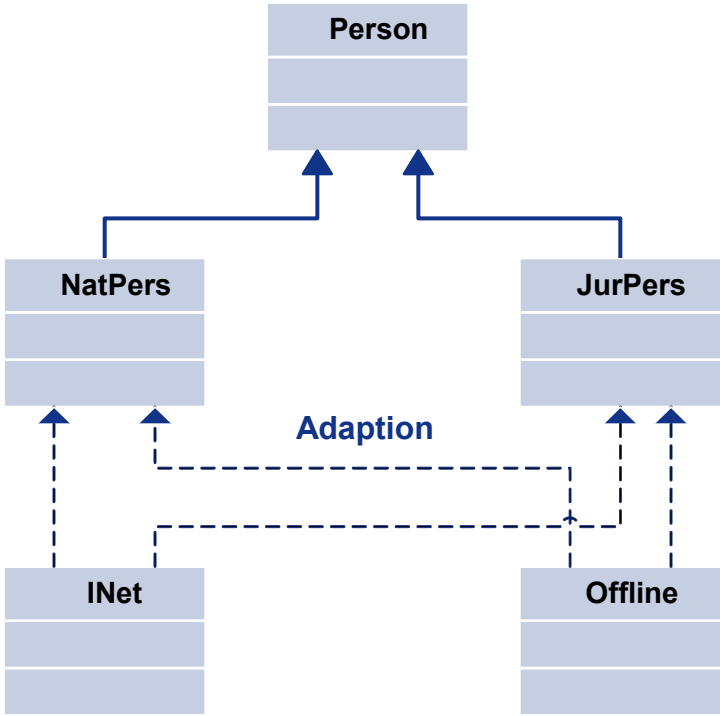


Explosion of extensions!

Allianz

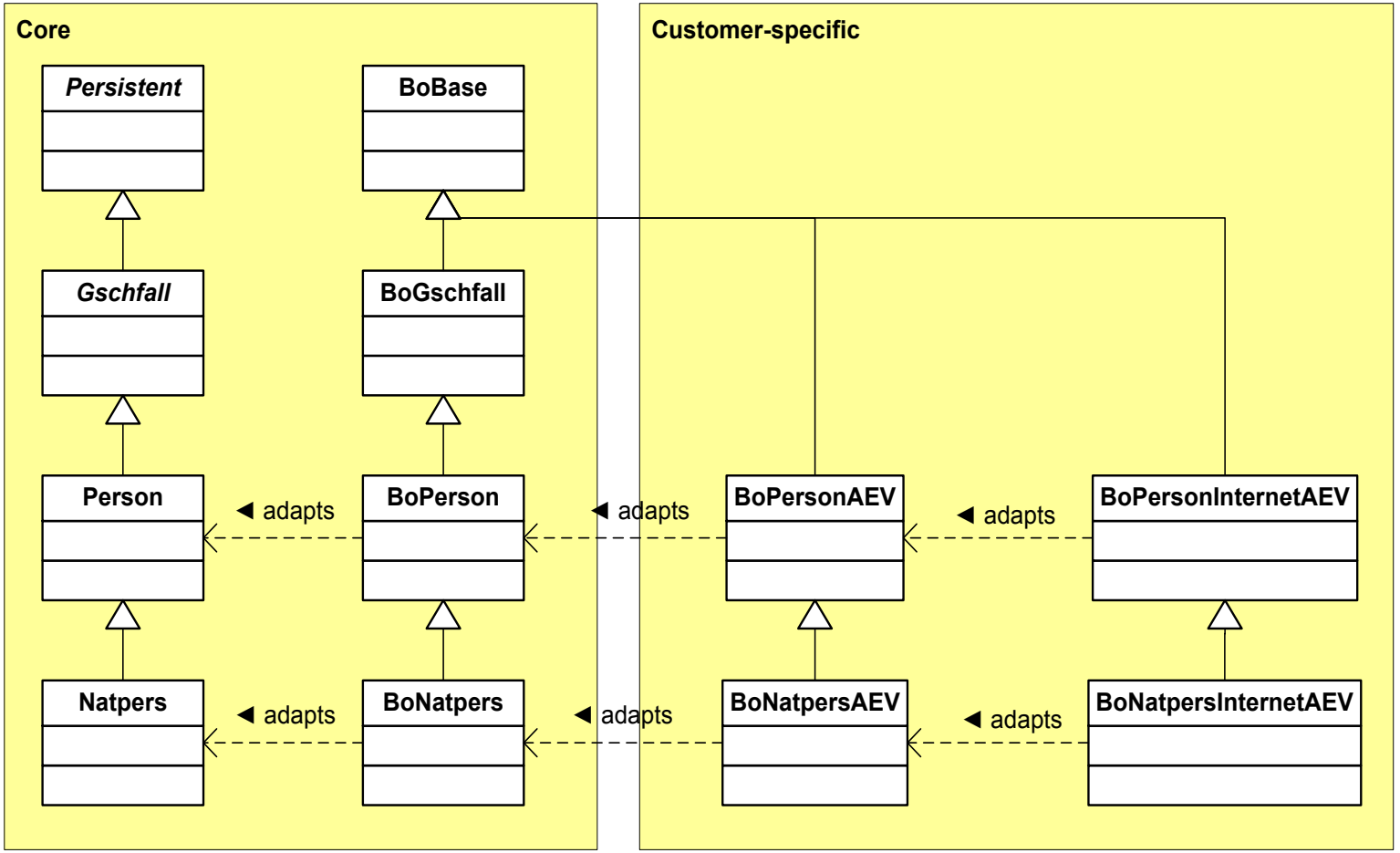
Adaption vs. inheritance: a better solution

- 1st aspect
- 2nd aspect
- …



Two mappings, but only one class is necessary in each case

Inheritance in the business layer (2)



Allianz

API vs. internal

- API = code that is kept stable after publication
  - Exception:
    Generated persistence layer is part of the API, but can be developed further in an incompatible manner due to DB model changes
- Internal code = private code that can be changed as required at any time
- Customer-specific sections may only use the API of the core
- Objective: transparent core updates for customers

Allianz

API vs. internal (2)

- Core plug-ins can use internal information of other core plug-ins
- Compiler settings:
  - Core: switch off warning “discouraged reference”
  - Customer: activate warning “discouraged reference”
- Guidelines for core development:
  - For each class and each method, you have to consider whether they belong in the API or are internal.
  - The API should be as small as possible and only as large as necessary

Quiz 3

Question 1

ABS is divided into several plug-ins, which represent the different functional aspects. What are the conventions when working with plug-ins in A3k?

a. Singleton option for plug-ins should be set
b. Plug-ins must be versioned
c. Use plug-in dependencies and not package imports
d. Should always be based on a template
e. Activator class should contain a constant PLUGIN_ID

Solution: a, c, e

Question 2

What is an adapter?

a. Is derived from BoBase
b. Created internally using an adapter factory which is registered for the adapter class
c. Application of the extension object pattern
d. Extension objects that can be queried at runtime from an adaptable object
e. Stored in the database
f. Dies with the adapted object

Question 3

What are the characteristics of a business object?

a. Stateless
b. Class names have prefix ‘Bo’
c. Derived from `BoBase`
d. Not managed by the `framework`
e. Know their `UnitOfWork` and the `A3kSession`

Solution: b, c, e

Question 4

Which object should be adapted is determined by the lifecycle and the functional associations. What are possible objects to be adapted?

a. Persistent object
b. Unit of work
c. Interface
d. Internal classes
e. Business object

Allianz

Question 5

Each adapter has a lifecycle. Which statements are correct?

a. For each access a new adapter object is created
b. Clean-up of adapters has to be done by developer
c. Lifecycle is managed by the framework
d. Adapter objects are created when accessed for the first time
e. Must be stored in the database
f. Obtain adapter via `getTypedAdapter()`

Solution: c, d, f

Allianz

Question 6

UnitOfWork mappings are used for registering adapter factories on a context-specific basis. Which two methods can be used in order to map objects?

a. mapAdapterFactories
b. mapLocalAdapter
c. mapCoreAdapter
d. mapSingleAdapter
e. mapStrategy

Solution: b, c, e

Question 7

Which of the following examples are correct, when using inheritence within the business logic layer in ABS?

a. `BoNatpers extends BoPerson`

b. `BoPersonCustomer extends BoPerson`

c. `BoJurpers extends BoPerson`

d. `BoExtendedPerson extends BoPerson`

Question 8

What are the differences between API code and internal code?

a. Internal code can be used by the customer
b. API code is published in the web
c. Internal code can be changed as required at any time
d. Extension points are always internal
e. API code is kept stable after publication

Solution: c, e

Allianz

Exercise 2.2.1: Create a business logic class

- Create your own business logic plug-in; create a business logic class and register an adapter
  - A new business logic class should be created in your own business logic plug-in. (The search logic will be implemented in this logic class in the following steps.)
  - The business logic class should be able to be tested using JUnit, for which a test fragment should be created.
  - A JUnit test has to show that the business logic class can be accessed within the person file as an adapter of Natpers.

Allianz

Business attributes

- Attributes of business objects are implemented as so-called **business attributes** or **BoAttributes**
- Attribute objects contain a value of a certain type
- Example:
  - `public class BoAdresse extends BoBase {`

```
public IAttributeWritable<String> getAttributeStreet() { …
public IAttributeWritable<Integer> getAttributePostalCode() { …
 …
```

- Depending on the functionality, they implement certain interfaces with corresponding methods
  - “Read”, “Write”, “Domain”, “Values”, “Range”, “Extendable”
- Corresponding interfaces are available for all combinations that make sense (see following table later on)

Allianz

Possible functions for business attributes

- **“Read”:** attribute can be read (applies to all): IAttributeRead<T>
  - Methods getValue(), isAvailable()
- **“Write”:** attribute can be edited: IAttributeWritable<T>
  - Methods setValue(T), validateValue(T value), isChangeable()
- **“Domain”:** attribute belongs to a domain: IAttributeReadDomain
  - Methods getDomain(), getValidDomain(), getDomainText()
- **“Values”:** value only from list of acceptable values: IAttributeReadValues<T>
  - Method getValues()
- **“Range”:** value range with upper/lower limit: IAttributeWritableRange<T>
  - Methods getMin(), getMax(), getStepSize()
- **“Extendable”:** can be extended for customer: IAttributeExtendable<T>

Allianz

BoAttribute interfaces

	Read	Write	Domain	Values	Range
IAttributeRead	■				
IAttributeWritable	■	■			
IAttributeReadDomain	■		■		
IAttributeWritableDomain	■	■	■		
IAttributeReadValues	■			■	
IAttributeWritableValue	■	■		■	
IAttributeWritableRange	■	■			■

Allianz

Implementation of BoAttributes

- Framework offers base classes:
  - For transient attributes (values with the lifespan of a business object)
    - `BoAttributeRead, BoAttributeWritable` etc.
  - For persistent attributes (values originating from a persistent object)
    - `BoAttributeReadPersistent, BoAttributeWritablePersistent` etc.
  - A2k attributes (only for A3k encapsulation in the core)
    - `BoAttributeReadA2k, BoAttributeWritableA2k`, etc.
  - For other cases: derivation of `BoAttributeReadBase,` etc.

Convention: derivations as inner class, within a business logic class

Allianz

Implementation of BoAttributes

- Business logic class: method name `getAttribute…`
- `BoAttribute` base classes from the Framework can often be used directly, e.g.:
  - `attributeStreet = new BoAttributeWritablePersistent<String>(getPersistent(), PMAdresse.getStrasseInfo());`
- If required, based on individual logic, e.g. value validation:
  - Internal class within the business logic class, private or protected
  - Derivation of `BoAttribute…`class, e.g. `BoAttributeWritablePersistent`

Allianz

Name conventions for business logic classes

- Current convention, with business attributes

```
public class BoXxx {

  private class BoAttributeStreet extends BoAttributeWritableBase<String>{
    …
  }

  private final BoAttributeStreet street;

  public IAttributeWritable<String> getAttributeStreet() {...}

}
```

Allianz

Exercise 2.2.2: First, simple business logic

- BoAttributes; simple business logic methods
  - The search should enable the use of two search criteria, first name and academic title. The business logic class is therefore to make a note of these criteria in business attributes.
  - Furthermore, the business logic class should have a reset method so that both search criteria can be deleted again easily.

Allianz

Exercise 2.2.3: Business logic to run a view query

- Repeat: Define view class, database query
  - A search method should return a list of natural persons whose surname matches that of the person currently opened in the file and their first name and academic degree (‘Akadgrad’) with the search criteria specified.
  - The list of hits should contain the columns first name, surname, academic title and day of birth.
  - The database query is also to work if only one, or neither of the two search criteria has been specified. Furthermore, it should enable the wildcard ‘*’ to be used for the first name.
  - `BoNatpersSearch` should remember the results list and provide access to it.

Allianz

Validation and access restrictions

ValidationManager

- Collects user input validation messages of a page (so all can be displayed at once)
- To be obtained via `UnitOfWork.getValidationManager()`
- To set validation messages:
  - Information messages (information, method: `putInformation`)
  - Error messages (exclamations, method: `putExclamation`)
  - A unique ID must be supplied (important for web clients)

Implementation of validation logic

- Attribute validation: In `BoAttribute`, i.e. overwrite the method `ValidationResult validateValueDefault(T value)`
- Page or action validation: in an own `validate…` method for the business logic class, with return type `ValidationResult`

Allianz

Validation and access restrictions (2)

Execution of the validation logic

- Validation methods in the business logic layer are to be explicitly called from the dialog layer
- In order to do this, implement validate methods in `DoPage` and `DoField`
- Only exception, with implicit validation:
  `BoAttribute` connected with `DoFieldAttribute`

Allianz

Exercise 2.2.4: Validation

- Implement validation logic
  - For the search criterion ‘first name’, only letters and ‘*’ can be used. This should be ensured using a validation method. The user should see an error message.

Allianz

Exercise 2.2.5: Another business attribute

- A read-only business attribute
  - The business logic class should have a third attribute which displays the family name of the person. It should not be changeable.

Allianz

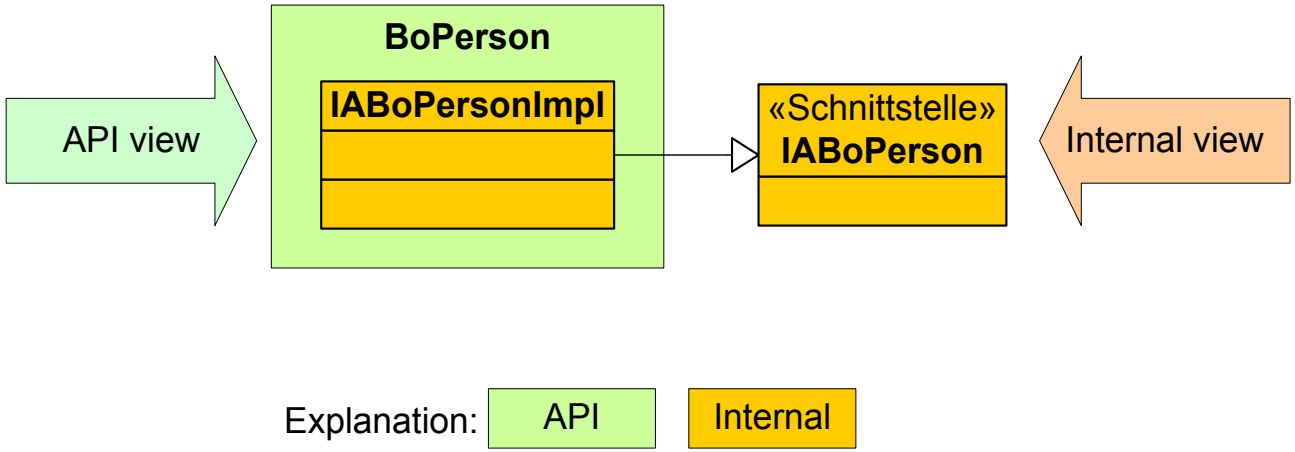
Integration adapter pattern

- Pattern for the core
- Objective: hide internal core methods for API objects
- Internal core interface inherits from IABase (package ending in ia.internal)
- Idea:
  - inner class in the API object implements the interface and extends IABaseImpl
  - inner class can be collected as an adapter for the API object
  - inner class has access to private elements of the surrounding class

Allianz

Integration adapter pattern

- Registration of integration adapters via the `IAFactory` class
- Examples: `UnitOfWork`, `BoPerson` and many more.



Allianz

Integration adapter example

- UnitOfWork: registering the Integration Adapter IAUnitOfWork

```
IAFactory.register(UnitOfWork.class,
        new IAFactory<UnitOfWork>(IAUnitOfWork.class) {
        // return Integration Adapter
        @Override
        protected IABase createAdapter(final UnitOfWork adaptable) {
                return adaptable.iaUnitOfWorkImpl;
        }
});
```

- UnitOfWorkMapper: call mapCoreAdapter via IAUnitOfWork implementation

```
UnitOfWork unitOfWork = ...;
IAUnitOfWork iaUnitOfWork =
        unitOfWork.getTypedAdapter(IAUnitOfWork.class);
iaUnitOfWork.mapCoreAdapter(Adaptable.class,
        new BoReflectionAdapterFactory(Adapter.class));
```

Allianz

Integration with A2k

- Access to A3kSession and A2kSession

```
A2kSession a2kSession = ...;
A3kSession a3kSession = a2kSession.getA3kSession();
```

```
A3kSession a3kSession = ...;
A2kSession a2kSessioon =
       a3kSession.getTypedAdapter(IAA3kSession.class).getA2kSession();
```

- Access to UnitOfWork and UobObjectManager

```
UobObjectManager om = ...;
UnitOfWork uow = (UnitOfWork)
       om.getAssociate(UoConst.ASSOCIATE_UNITOFWORK);
// for typed access from a2k business logic to a3k business logic
// the associate mechanism is used
```

```
UnitOfWork uow = ...;
UobObjectManager om =
       uow.getTypedAdapter(IAUnitOfWork.class).getObjectManager();
```

Allianz

Exercise 2.2.6: Using Integration Adapters

Correcting the mapping in our existing UnitOfWorkMapper class

- The plugin at.allianz.gfb.core.training.person is part of the core. Change the mapLocalAdapter call (for customer mappings only) in the UnitOfWorkMapper of exercise 2.2.1 to mapCoreAdapter using the correct integration adapter interface.

Quiz 4

Question 1

Business objects can have different attributes – BoAttributes. A BoAttribute can have different functions. What are possible functional aspects?

a. New
b. Read
c. Write
d. Range
e. Domain
f. Extendable
g. Visible
h. Transient

Solution: b, c, d, e, f

Question 2

A business attribute can represent different attributes. What are possible types?

a. Transient attributes
b. Application attributes
c. A2k attributes
d. Persistent attributes
e. Table attributes

Question 3

How should business attributes be implemented?

a. As extension
b. As interface
c. As internal class, within the business logic class
d. As static class

Solution: c

Question 4

Within A3k the validation manager concept is used for error handling. Which methods can be used to put validation messages?

a. putMessage
b. putExclamation
c. putInformation
d. putValidation
e. putInstruction

Solution: b, c, e

Allianz

3

Allianz

ABS extension concepts – configuration

Objective: change core behaviour to implement customer requests

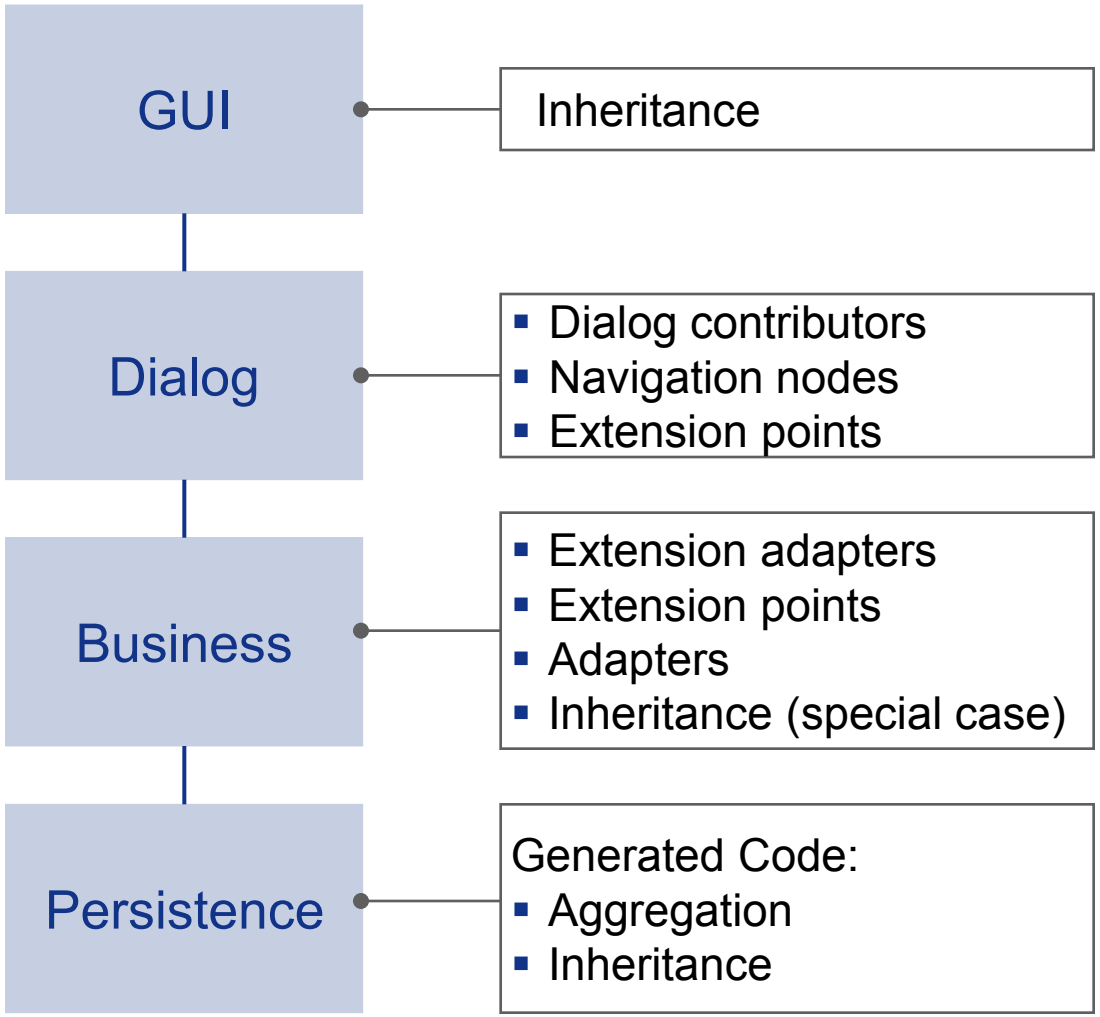
Two options: using parameters or program code

- Parameterisation
  - Alternative scenarios coded in the core that can be selected using a switch
  - Possible implementation via business foundation tables, e.g. TFirmendaten or configuration switch

Allianz

ABS extension concepts – programming

Core code must not be changed by the customer!



Allianz

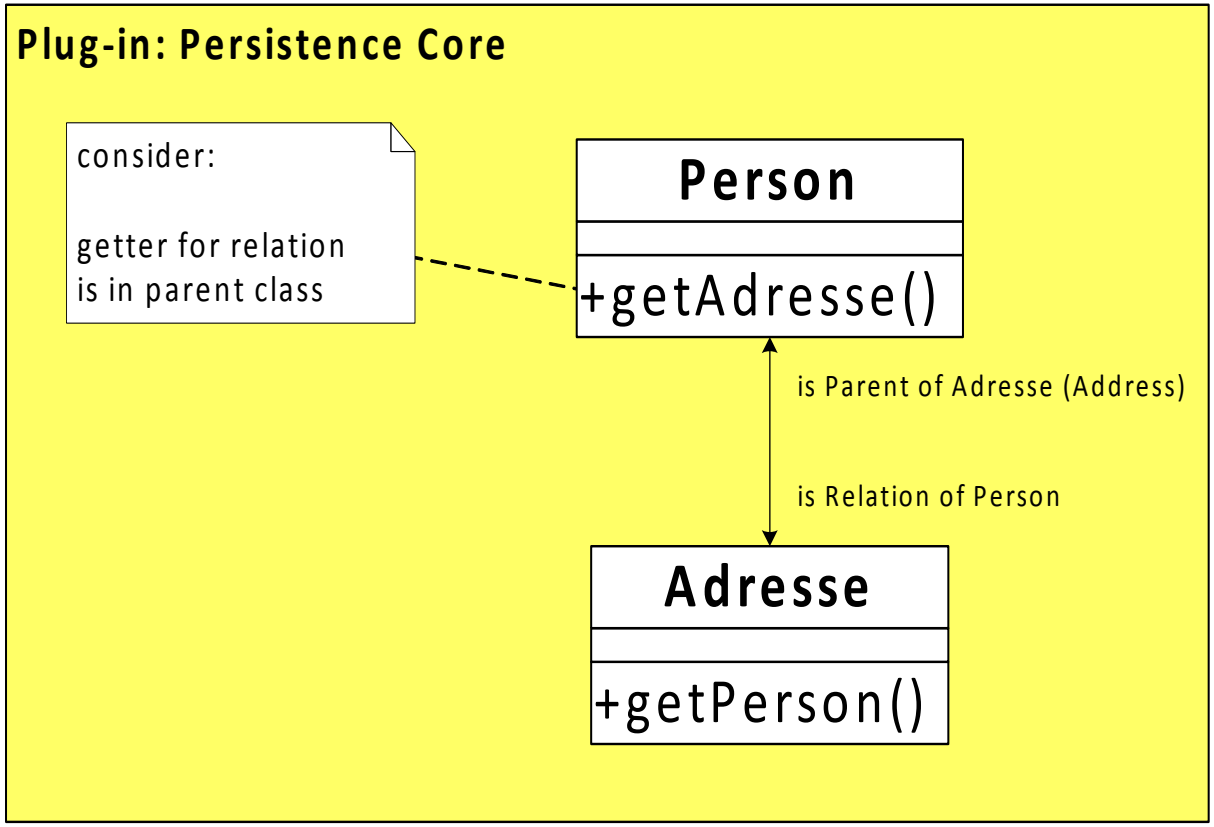
3

Allianz

Data model extensions

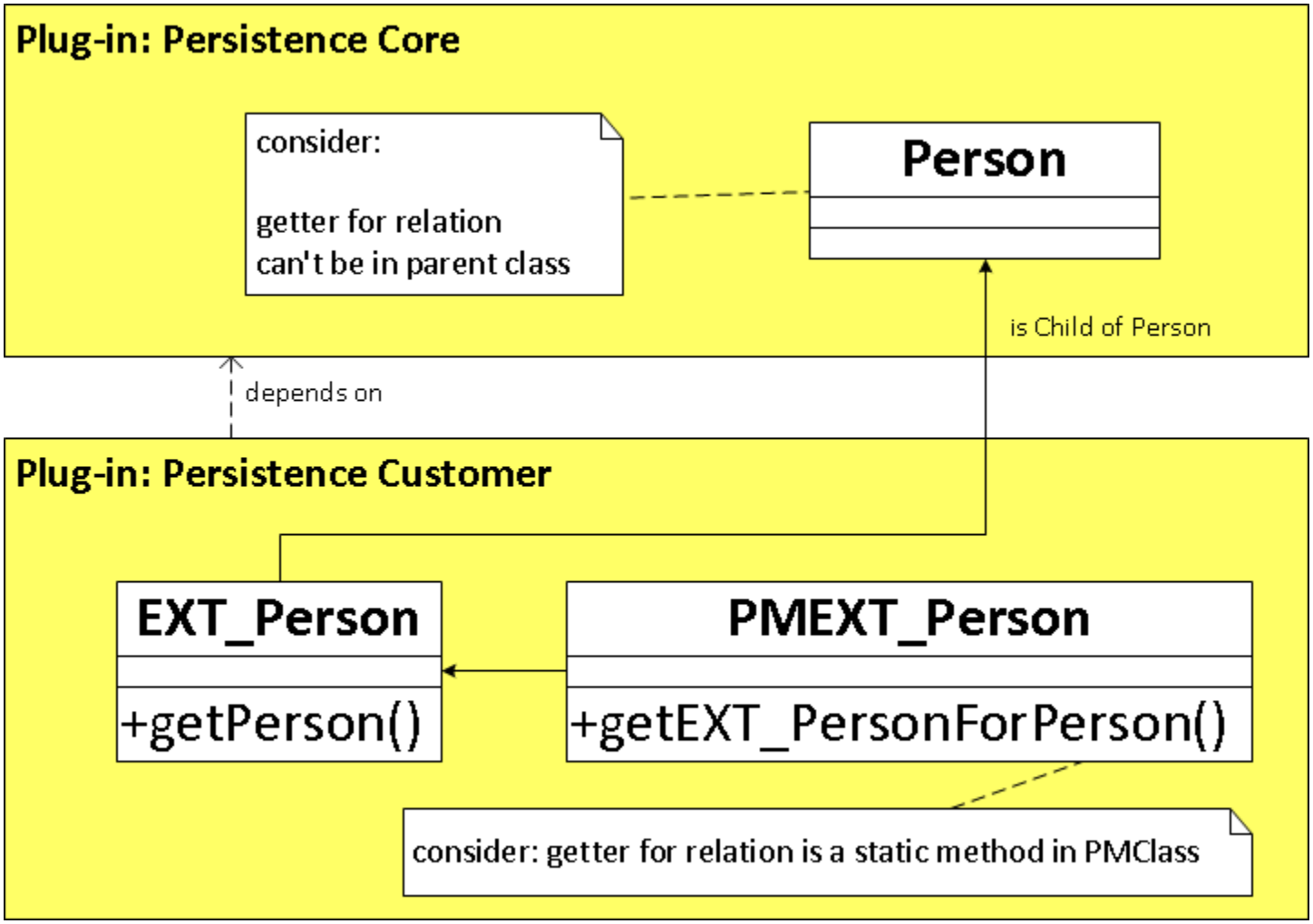
- Customer-specific extension of the data model is performed by way of additional tables
  - Core tables remain unchanged
  - An extension table is a child object which references a core table, i.e. remains an “attachment“
  - Customer-specific root objects are also possible
- Customer-specific table/persistent objects can be identified by a name prefix
  - E.g. AzA: `AT1_Natpers` contains AzA-specific extensions for `Natpers`
  - E.g. AzD: `TDE1_Natpers` contains AzD-specific extensions for `Natpers`
- Domains can also be subject to customer-specific extensions
  - Generated constant interfaces derive from core interfaces

Persistence layer – relations (core)



Allianz

Persistence layer extensions – relations over plug-in borders



Allianz

DataContainer extensions

- In order to add columns to an existing DataContainer, it can be extended dynamically with a DataContainerExtension
  - Create a subclass of `DataContainerExtension`
  - Define the additional columns there
  - Use the DataContainer method `getExtension` to access the extension
  - Never instantiate the extension class directly!

Allianz

Example – Definition of a DataContainer extension

- Extension of a DataContainer

```
public class DcExtPerson extends
             DataContainerExtension<DcPerson> {

    private final Column<Integer> numberOfChildren;

    public DcExtPerson(DcPerson dataContainer) {
        super(dataContainer);
        numberOfChildren = dataContainer.createColumn(Integer.class);
    }

    public Column<Integer> getNumberOfChildren() {
        return numberOfChildren;
    }
}
```

- Access to the extension

```
DcPerson dc = …;
DcExtPerson dcExt = dc.getExtension(DcExtPerson.class);
```

Allianz

Exercise 3.1.1: DataContainer extension

- DataContainerExtension
  - A DataContainer extension for the `DataContainer` from exercise 2.1.3 should be created.
  - A new column, the department name of the employee, should be added.
  - Fill in some data and display the entire DC content on the console.

Allianz

3

Allianz

Business logic – extension capability

Concept

- Only a few generic extension possibilities
- Extension possibilities need to be defined and programmed
  - Protects logic from incompatible changes made by customer
  - Change requirements via functional design teams

Examples

- Writing process slips or scheduled tasks
- Adding object specializations, e.g. order types
- Points for initialisation

Allianz

Plug-in component structure for extensions

- Core code must not be modified
  → Customer code in separate plug-ins
- Proven to be practical:
  - One customer extension plug-in per core plug-in which needs customisation; dependency to the core plug-in
  - Similar plug-in structure for customer plug-ins as for core plug-ins

Allianz

Exercise 3.2.1: Extension business logic class

- Creating an extension business logic class
  - In a new, customer-specific plug-in and a business logic class that adds customer-specific functionality to the existing BoNatpersSearch business logic class should be created.
    (An extended search logic will be implemented in this logic class in the following steps.)

Allianz

Exercise 3.2.2: Adding extended business logic

- Using a DataContainerExtension
  - The search by person should provide an additional column that includes the age of the person.

Allianz

Business logic – generic extension hooks

The business logic layer contains a few generic hooks for customer-specific logic.

Delegates and listeners can get registered:

- For `UnitOfWork: IUnitOfWorkDelegate`
  - Contains the lifecycle methods `onOpen, onSetWritable, onSave, postSave, onClose, getRight`
  - Must be registered at the extension point “unitOfWorkDelegate”
- For `Persistent: IPersistentDelegate`
  - Contains the lifecycle methods `onSave, postSave`
  - Must be registered at the extension point “persistentDelegate”
- For BoBase: IBoListener
  - Contains the lifecycle method `onInit`
  - Has to be stateless
  - Must be registered programmatically (in the UnitOfWorkMapper class)

Allianz

Calling customer-specific code in the core

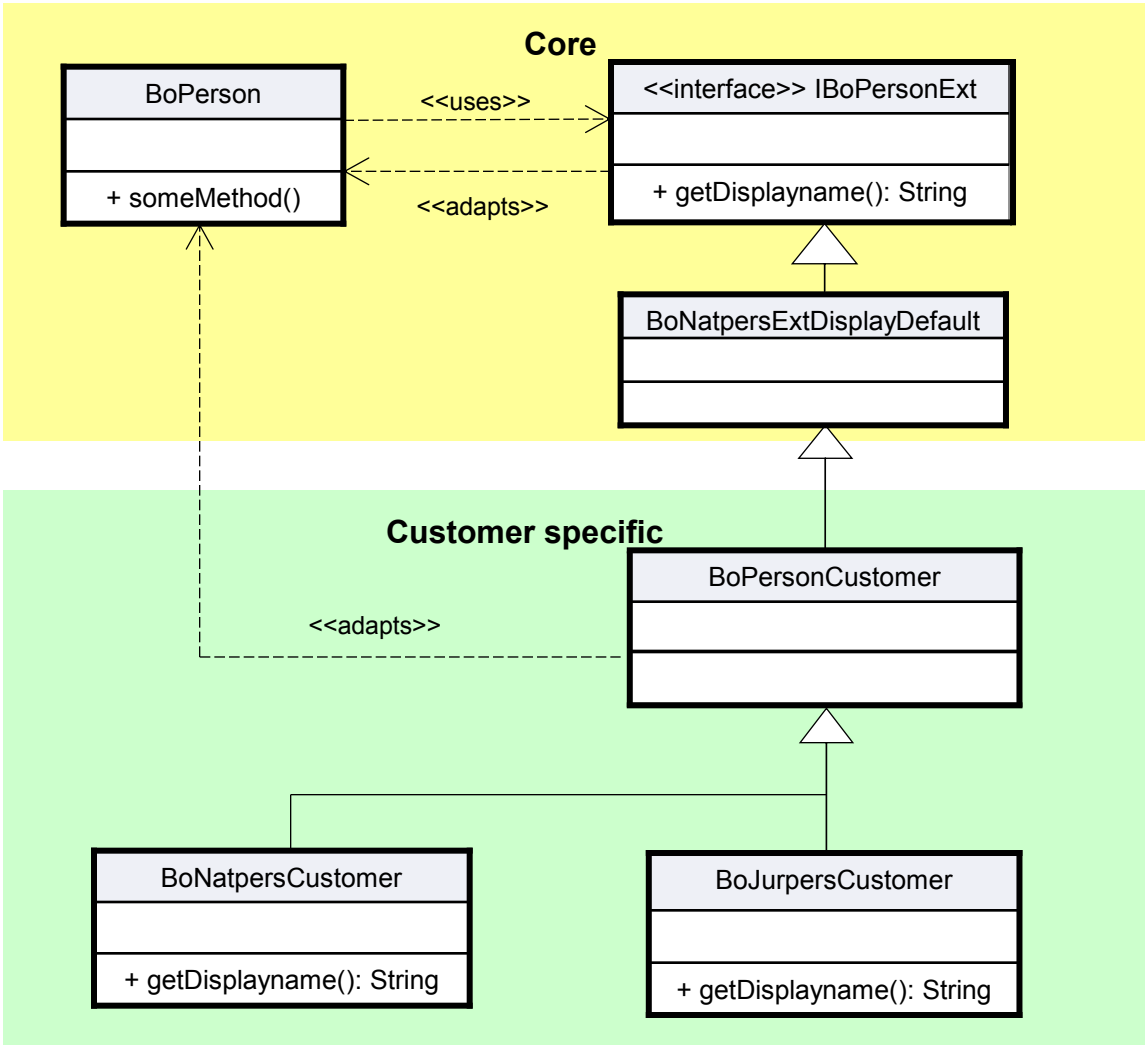
Eclipse extension points

- Known Eclipse mechanism
- An unlimited number of extensions can be registered at an Extension Point (but a limit can be specified in the Extension Point definition)

Extension adapter

- Extensions based on adapter and UnitOfWork mapping
- If 0 to 1 extensions are desired
- For very granular extensions
- Can be used for internal core and customer-specific extensions

Extension adapters



Internal:

Core default implementation

→ `mapStrategy()`

API:

Customer extension

→ `mapStrategy()`

Allianz

Extension adapters – step 1 (core)

- Core defines an extension interface
- Naming convention: `I<BusinessObjectName>Ext<OptionalSuffix>`

```
public interface IBoPersonExtDisplay {

    String getDisplayname();

}
```

Allianz

Extension adapters – step 2 (core)

- Core implements a default extension class

```
public class BoNatpersExtDisplayDefault extends BoBase
        implements IBoPersonExtDisplay {

    public BoNatpersExtDisplayDefault(BoNatpers boNatpers) {
        super(boNatpers);
    }

    public String getDisplayname() {
        BoNatpers boNatpers = getBusinessObject();
        return …
    }

}
```

Allianz

Extension adapters – step 3 (core)

- Mapping of the default core implementation to the extension interface

```
public class UowMapperPersonCore implements IUnitOfWorkMapper {

    @Override
    public void map(UnitOfWork unitOfWork) {
        unitOfWork.mapStrategy(BoNatpers.class,
                new BoReflectionAdapterFactory(
                BoNatpersExtDisplayDefault.class,
                IBoPersonExtDisplay.class));
    }
}
```

Allianz

Extension adapters – step 4 (core)

- Use of the extension interface in the core

```
public class BoPerson extends BoBase {

    public String someMethod() {

        // Fetch adapter from the object itself
        IBoPersonExtDisplay ext =
            this.getTypedAdapter(IBoPersonExtDisplay.class);
        // In case that the customer did not register something:
        // Core default adapter gets returned

        // In case that the customer registered an own implementation:
        // The adapter of the customer gets returned

        String displayname = ext.getDisplayname();
    }

}
```

Allianz

Extension adapters – step 5 (customer)

- Customer implements an extension class

```
public class BoNatpersExtDisplayCustomer extends
        BoNatpersExtDisplayDefault {

    public BoNatpersExtDisplayCustomer(BoNatpers boNatpers) {
        super(boNatpers);
    }

    @Override
    public String getDisplayname() {
        BoNatpers boNatpers = getBusinessObject();
        return boNatpers.getAttributeVorname().getValue() + " "
            + boNatpers.getAttributeFamname().getValue();
    }
}
```

It is recommended to extend the default extension of the core

Allianz

Extension adapters – step 6 (customer)

- Mapping of the customer-specific implementation to the extension interface

```
public class UowMapperPersonCustomer implements IUnitOfWorkMapper {

    @Override
    public void map(UnitOfWork unitOfWork) {
        unitOfWork.mapStrategy(BoNatpers.class,
                new BoReflectionAdapterFactory(
                BoNatpersExtDisplayCustomer.class,
                IBoPersonExtDisplay.class));
    }
}
```

NB: This presupposes that the customer mapping has priority against the mapping of the core default implementation (larger plug-in distance)

Allianz

Exercise 3.2.3: Changing a business logic procedure using extension adapters

- Implement and register extension adapters
  - The claims workflow contains an extension link, which allows exactly one extension to be incorporated. The documentation on the `IBoSchadenworkflowExt` extension adapter can be found in the Programmer's Guide within the ABS core documentation (in the Eclipse help system).
    - The functionality of this extension adapter should be demonstrated using a simple console output via `System.out`.

Allianz

Exercise 3.2.4: Defining and using extension adapters

- Define an extension adapter interface; implement and register an extension adapter
  - The customer should be able to configure the maximum size of the view performed by BoNatpersSearch
    - Step 1, core system: Introduce an extension adapter IBoNatpersSearchExt.
    - Step 2, customer code: Write a customer-specific implementation of this extension adapter.

Allianz

Extensible business attributes

- Attributes can be extended in two manners: by listeners or by extensions.
  - **Listeners** can be registered for **validation** and **setting** aspects (of the attribute value). These are **additive** and multiple listeners can be registered to any writable business attribute.
  - **Extensions** can be registered to the attributes for **all other aspects**. The extension **replaces** the attribute logic. The last registered extension will take effect (chain of extensions).
- Attribute listeners or extensions have to be registered in a business logic adapter listener (`IBoListener`). A business logic adapter listener can be registered to a business object class in a `UnitOfWorkMapper`.

Allianz

Extensible business attributes (2)

- Customer-specific extensions which are possible for all business attributes:
  - `IAttributeWriteListener`: Methods `setValue`, `validateValue`
    Registration via `addWriteListener`
  - `IAttributeReadExtension`: `isAvailable`
    `IAttributeWriteExtension`: `isChangeable`
    `IAttributeDomainExtension`: `getDomain`, `getValidDomain`
    `IAttributeValuesExtension`: `getValues`
    Registration via `registerExtension`
- Only allowed for attributes which implement the marker interface `IAttributeExtendable`:
  - `IAttributeValueExtension`: Method `getValue`
    Registration via `registerExtension`
- Several extension base classes simplify the implementation: `BoAttribute*ExtensionBase`, e. g. `BoAttributeRead-ExtensionBase`, `BoAttributeWriteDomainExtensionBase`

Allianz

Extensible business attributes – extension procedure

- To register an extension we need a hook: `IBoListener`
- `onInit` should contain the necessary code to add a business attribute extension
- Can be registered through method `registerBoListener` on the UnitOfWork
- This should be done in the UnitOfWorkMapper after the registration of the adapters

Allianz

Business attributes – step 1 (core)

- Core has already defined an extendable business attribute
  Example (see class `BoJurpersUsecaseAttribute`):

```
private final BoAttributeWritablePersistentExtendable<String>
        attributeCompanySeal = new
        BoAttributeWritablePersistentExtendable<String>(persistent,
        PMJurpers.getFirmenstempelInfo());

…

// Getter
public final IAttributeWritableExtendable<String>
        getAttributeCompanySeal() {
    return attributeCompanySeal;
}
```

Allianz

Business attributes – step 2 (customer)

- Implement a listener or an extension

```
public class BoAttributeCompanySealListener implements
        IAttributeWriteListener<String> {

    @Override
    public ValidationResult validateValue(String value) {
        // Customer implementation
    }
    …
}

public class BoAttributeCompanySealExtension extends
        BoAttributeWriteExtensionBase<String> {

    @Override
    public boolean isChangeable() {
        // Customer implementation
    }
    …
}
```

Allianz

Business attributes – step 3 (customer)

- Register the attribute listener or extension in an `IBoListener`

```
public class BoJurpersUsecaseAttributeListener implements
         IBoListener<BoJurpersUsecaseAttribute> {

    @Override
    public void onInit(BoJurpersUsecaseAttribute businessObject) {

        BoAttributeCompanySealListener listener =
             new BoAttributeCompanySealListener();
        businessObject.getAttributeCompanySeal()
             .addWriteListener(listener);

        BoAttributeCompanySealExtension extension =
             new BoAttributeCompanySealExtension();
        businessObject.getAttributeCompanySeal()
             .registerExtension(extension);
    }
}
```

Allianz

Business attributes – step 4 (customer)

- Register the `IBoListener`

```
public class UowMapperUsecaseAttributeExtension implements
        IUnitOfWorkMapper {

    @Override
    public void map(UnitOfWork unitOfWork) {
        unitOfWork.registerBoListener(BoJurpersUsecaseAttribute.class,
                new BoJurpersUsecaseAttributeListener());
    }
}
```

Quiz 5

Question 1

Extending the persistence layer can affect the data model. How customer-specific extensions of the data model are performed?

a. Manipulating the core tables
b. Extending the data model is impossible
c. Additional tables
d. Replacing the core tables

Solution: c

Allianz

Question 2

Data container can be extended by DataContainerExtensions. How the data container extension can be accessed?

a. Instantiate the data container extension class directly
b. Via the method getExtension()
c. Store an instance of the data container extension in the base data container class.
d. Via the method accessDataContainerExtension()

Solution: b

Allianz

Question 3

Extensions should be implemented in a plug-in component structure. Do you remember best practices regarding to this?

a. Customer code should not have a similar plug-in structure as the core.
b. The core code must be modified.
c. A strict separation between core and customer code is not desired.
d. The core code must not be modified.
e. There should be one customer extension plug-in per core plug-in which needs to be extended.

Solution: d, e

Question 4

A3k provides two mechanisms to call customer specific code from the core. Please designate them?

a. Extension points
b. Persistent delegates
c. Integration adapters
d. Extension adapters

Question 5

What are characteristics of the extension adapter pattern within A3k?

a. Used to call core code in the customer-specific extension
b. Based on the adapter concept
c. Used, if more than 1 extension is desired
d. Used for very granular extensions
e. Used to call customer-specific code in the core
f. Used, if 0 or 1 extension are desired

Solution: b, d, e, f

Question 6

Extendable business attributes can be extended in two manners. Do you remember them?

a. Attachment
b. Listener
c. Agent
d. Extension
e. Contributor
f. Add-on

Question 7

For which aspects of an business attribute a listener can be registered?

a. Validation
b. Setting
c. Visibility
d. Extendibility

Question 8

For which aspects of an business attribute an extension can be registered?

a. Getting
b. Setting
c. Changeability
d. Availability
e. Validation

Solution: a, c, d

Allianz

Exercise 3.2.5: Business attribute extension

- Implement a business attribute extension.

  Write an extension in our customer plug-in which makes the attribute ‘AcadDegree’ of our ‘core’ class BoNatpersSearch read-only.

Allianz

metafinanz
we ensure success

*