PSE Ausarbeitung

Phillip Wo
Benjamin Moser
Daniel Sommer

March 7, 2019

Contents

I Pattern im Projekt **4**

1 Layers Pattern **4**
1.1 Erkläre die Funktionsweise + Skizze 4
1.1.1 3 Schichten Architektur: 4

2 Data Access Object (DAO) Pattern **5**
2.1 Erkläre die Funktion + Skizze 5
2.2 Nenne die Konsequenzen der Anwendung 6

3 Service Layer Pattern (auch Session Fassade - in unserem Projekt im Domain Layer **6**
3.1 Erkläre die Funktion + Skizze 6
3.2 Nenne die Konsequenzen der Anwendung 9

4 Model-View-Controller (MVC) Pattern **9**
4.1 Erkläre die Funktion + Skizze 9
4.1.1 Model 10
4.1.2 View 10
4.1.3 Controller 10

5 Front Controller **10**
5.1 Erkläre die Funktion + Skizze 10
5.2 Servlet 10
5.2.1 Java Server Faces (JSF) 11
5.3 Nenne die Konsequenzen der Anwendung 13

6 View Helper (*/src/main/java/at/fhj/swd/psoe/web/)** **13**
6.1 Erkläre die Funktion + Skizze 13
6.2 Nenne die Konsequenzen der Anwendung 13

7 Dependency Injection (CDI-Framework -> eingebunden im *./pom.xml*) **14**
7.1 Erkläre die Funktion + Skizze 14
7.2 Nenne die Konsequenzen der Anwendung 15

8 Data Transfer Object (DTO) Pattern **15**
8.1 Erkläre die Funktion (Skizze - ein Grund für DTO) 15
8.2 Konsequenzen der Anwendung 17

II Exception Handling **18**

9 Checked und Runtime Exceptions in Java **18**
9.1 Checked Exceptions (z.B. SQL-Exception) 18
9.2 Unchecked Exceptions (z.B. NullPointerException) 18

10 Best Practice Beispiele beim Einsatz von Exceptions **18**

11 Exception Handling Anti Pattern **19**

12 Destructive Wrapping im Service Layer **19**

13 Exception Testing **20**

III Allgemeines & Config **21**

14 Logging **21**
14.0.1 Vorteile Logging mittels Framework (z.B.: log4j) 21

15 Annotationen **21**
15.1 Annotationen - Details 22

16 Konfigurationsdateien **23**
16.1 persistence.xml 23
16.2 web.xml 24
16.3 pom.xml 26
16.3.1 Aufbau pom.xml 26
16.4 standalone-psoe.xml 26
16.5 log4j.properties 28

17 Fehler im Projekt **28**
17.1 Return null 28
17.2 Exception nicht gefangen 28
17.3 Destructive Wrapping - Logging fehlt - Information geht verloren 28
17.4 Logger mit falschem Parameter 29
17.5 Fehlendes Exception Handling 29
17.6 Exception werfen und gleich wieder fangen 30

18 Tests **31**
18.1 Testpyramide 32
18.2 Unit 32
18.3 Integration Tests 33
18.4 GUI-Test 33
18.4.1 Page Object Pattern 33
18.5 Page Object Pattern lt. Zahnlücke 34
18.6 Beispiel aus dem Projekt 35
18.6.1 Integration GUI Test mit Selenium 35
18.6.2 Durch Selenium getestetes Page Objekt 36

19 Toni FRAAGNAA **37**

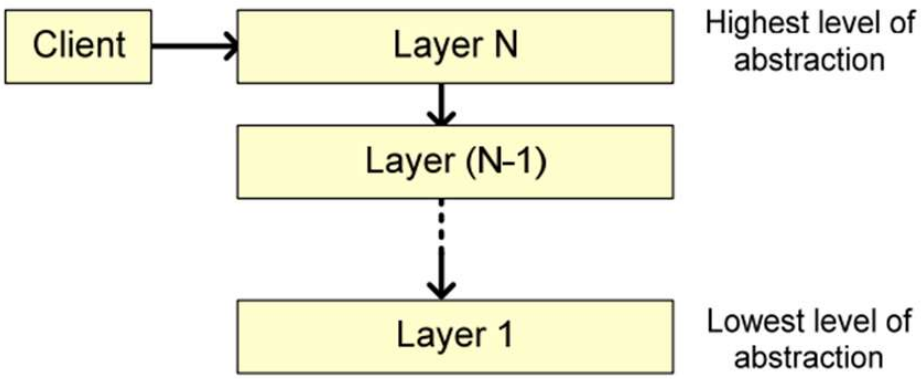
20 Frageart Prüfung **39**

Part I
Pattern im Projekt

1 Layers Pattern

1.1 Erkläre die Funktionsweise + Skizze

- Client schickt eine Anfrage an Layer N
- Layer N reicht da er nicht vollständig alleine beantworten kann, Anfragen an darunterliegenden Layer weiter
- Eine Anfrage kann bei Bedarf auch in mehrere Anfragen an darunterliegende Layer geteilt werden
- dies wird immer weiter fortgesetzt bis Layer 1 erreicht ist
- dabei gehen Abhängigkeiten nur von oben nach unten



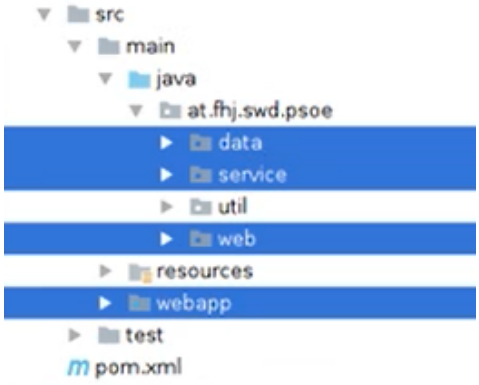
1.1.1 3 Schichten Architektur:

- Data Source Layer (data): Zugriff auf Daten, kümmert sich um Kommunikation mit anderen Systemen (z.B.: Datenbank)
  - enthält Entities –> Java Repräsentation vom DB Entity
    - im Projekt wurde eine AbstractEntity erstellt, welche die id managed
  - beinhaltet DAO und DAOImpl » DocumentDAO, DocumentlibraryDAO
    - damit man auf die Entities zugreifen kann.
    - um die DB zu abstrahieren.
    - enthält Methoden mit denen auf die DB zugegriffen wird
    - eine DAOException kontrolliert den Input
      - der EntityManager Aufruf in DAOImpl befindet sich innerhalb eines Try Blocks
      - im catch wird der Cause in die DaoException gewrapped

- Domain Layer(service): enthält Business Logik (Berechnungen, Datenvalidierung, ...)
  - beinhaltet
    - **Service Layer Pattern** (aka Session Fassade - siehe 3)
    - DTO » DocumentDTO
    - Mapper » DocumentMapper

```
public static Document toEntity(DocumentDTO documentDTO, Document
↪  document){};
public static DocumentDTO toDTO(Document document){};
```

- Presentation Layer(web): serverseitig, kümmert sich um Benutzerinteraktion
  - Controller (ViewHelper) » DocumentController, DocumentListController
  - View (WebApp)



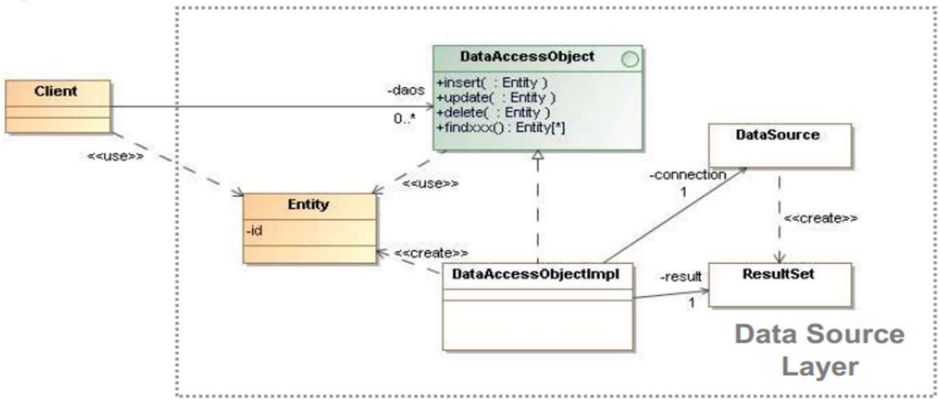
2 Data Access Object (DAO) Pattern

Befindet sich im Projekt in data und damit innerhalb des Data Layer.

2.1 Erkläre die Funktion + Skizze

- Client erstellt ein DAO Object und kann nach Entitäten suchen, einfügen, löschen, etc.
- das DAO selbst soll keine spezifischen Elemente enthalten (Entity Manager, SQL Exception -> stattdessen DAOException)
- dadurch entsteht eine Kapselung bei der die DAOImpl ohne den Client zu verändern ausgetauscht werden kann

```
@ApplicationScoped
public class DocumentDAOImpl implements DocumentDAO, Serializable {
    private static final long serialVersionUID = 1L;
    private static final Logger logger =
    ↪  LoggerFactory.getLogger(DocumentDAOImpl.class);
```




```
    @PersistenceContext
    private EntityManager entityMangaer;

    @Override
    public List<Document> findByCommunity(Community community) {...}

    @Override
    public List<Document> findByUser(User user) {...}

    @Override
    public void insert(Document document) {...}

    @Override
    public void delete(Document document) {...}

    @Override
    public Document findById(Long id) {...}
}
```

2.2 Nenne die Konsequenzen der Anwendung

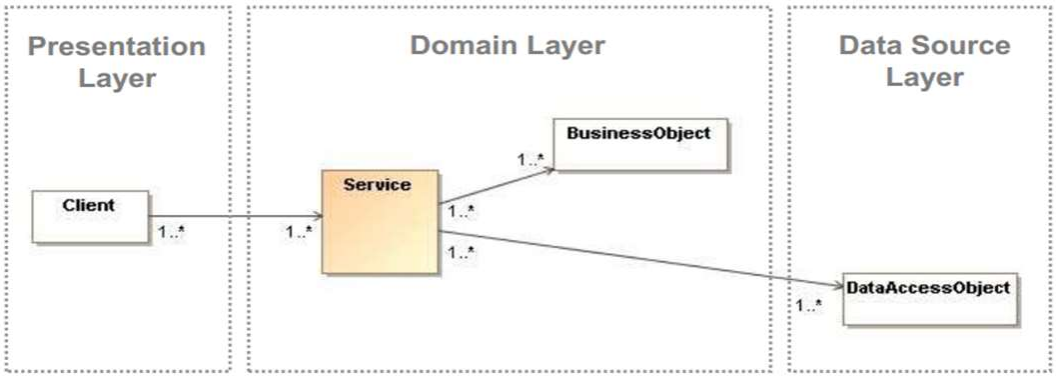
- Zugriff auf persistenten Speicher wird abstrahiert
- Details des Speichers werden versteckt
- ermöglicht einheitlichen Zugriff auf Daten
- entkoppelt Implementierung von Persistierung (Datenbank,...)
- ermöglicht Objektorientierte Ansicht des Speichers

3 Service Layer Pattern (auch Session Fassade - in unserem Projekt im Domain Layer

3.1 Erkläre die Funktion + Skizze

- Der Service Layer (Ordner "'service"' im Projekt) delegiert auf die Business Logik (Zeile 68 community.setDocumentlibrary) und zum DAO (z.B. Zeile 66)

- Bei wenig Logik wird zumindest Transaktions (Zeile 40), Error (ab Zeile 42) und Validierungshandling (ab Zeile 23) im Service erledigt




```
1   @Local(DocumentService.class)
2   @Remote(DocumentServiceRemote.class)
3   @Stateless
4   public class DocumentServiceImpl implements DocumentService,
    ↪  DocumentServiceRemote, Serializable {
5       private static final long serialVersionUID = -1L;
6       private static final Logger logger =
        ↪  LoggerFactory.getLogger(DocumentServiceImpl.class);
7
8       @Inject
9       private DocumentDAO documentDAO;
10      @Inject
11      private DocumentlibraryDAO documentlibraryDAO;
12      @Inject
13      private CommunityDAO communityDAO;
14      @Inject
15      private UserDAO userDAO;
16      @Inject
17      private MessageDAO messageDAO;
18      @Override
19      public DocumentDTO addDocument(Long communityID, String userID, byte[]
        ↪  data, String filename) {
20          Document addedDocument;
```

```
21              User user;
22
23              // Validierungshandling gefolgt von Error Handling
24              try {
25                  if (communityID <= 0) throw new
                    ↪  IllegalArgumentException("community must not be empty");
26                  if (userID == null) throw new IllegalArgumentException("user must
                    ↪  not be empty");
27                  if (data == null) throw new IllegalArgumentException("uploaded
                    ↪  file must not be empty");
28                  if (filename == null) throw new
                    ↪  IllegalArgumentException("filename must not be empty");
29
30                  Documentlibrary documentlibrary =
                    ↪  documentlibraryDAO.findByCommunityId(communityID);
31
32                  //create a document library, if there isn't already one in the
                    ↪  database
33                  if (documentlibrary == null) {
34                      documentlibrary = addDocumentlibrary(communityID);
35                  }
36
37                  user = userDAO.getByUserId(userID);
38
39                  addedDocument = new Document(documentlibrary, user, filename,
                    ↪  data);
40                  documentDAO.insert(addedDocument); // Transaktionshandling
41                  logger.info(String.format("Document %s saved in database",
                    ↪  filename));
42              // Error Handling
43              } catch (IllegalArgumentException iaex) {
44                  String errorMsg = "Uploading file failed (illegal argument)";
45                  logger.error(errorMsg, iaex);
46                  throw new ServiceException(errorMsg);
47
48              } catch (Exception ex) {
49                  String errorMsg = String.format("Uploading file %s failed.",
                    ↪  filename);
50                  logger.error(errorMsg, ex);
51                  throw new ServiceException(errorMsg);
52              }
53
54              String msgText = "Uploaded Document " + filename + " by user " +
                ↪  user.getUserId();
55              addMessageToStream(communityID, user, msgText, addedDocument);
56              return DocumentMapper.toDTO(addedDocument);
57          }
58
59
60          private void addMessageToStream(Long communityID, User user, String text,
            ↪  Document document) {...}
61
62          private Documentlibrary addDocumentlibrary(Long communityID) {
```

```
63              logger.info("Create missing documentlibrary");
64              Community community;
65              Documentlibrary documentlibrary = new Documentlibrary();
66              documentlibraryDAO.insert(documentlibrary); // Delegation zum DAO
67              community = communityDAO.findById(communityID); // Delegation zum
                ↪  DAO
68              community.setDocumentlibrary(documentlibrary); // Delegation zur
                ↪  Business Logik (Entity)
69              communityDAO.update(community); // Delegation zum DAO
70              return documentlibrary;
71          }
72
73          @Override
74          public List<DocumentDTO> getDocumentsFromCommunity(Long communityID)
            ↪  {...}
75
76          @Override
77          public List<DocumentDTO> getDocumentsFromUser(String userID) {...}
78
79          @Override
80          public void removeDocument(Long documentID) {...}
81
82          @Override
83          public DocumentDTO getDocumentById(Long documentID) {...}
84      }
```

3.2 Nenne die Konsequenzen der Anwendung

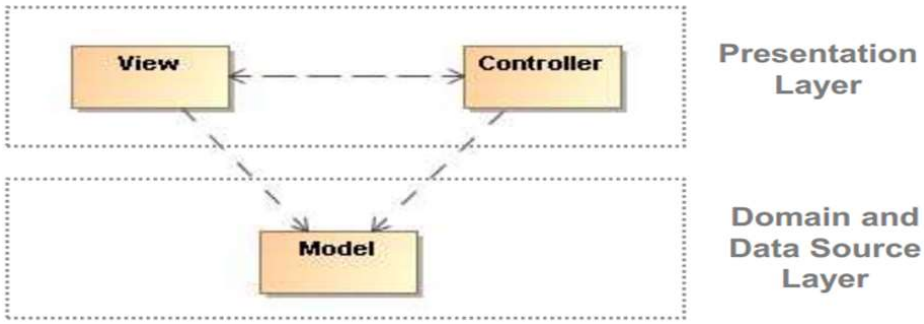
- Reduzierung der Abhängigkeiten zwischen Presentation und Domain Layer
- Zentralisiertes Sicherheits und Transaktionshandling
- verbirgt vor Client Komplexität der Business Logik
- stellt Client ein grobkörniges Interface zur Verfügung
- gut für Remote Aufrufe geeignet (weniger Aufrufe)

4 Model-View-Controller (MVC) Pattern

4.1 Erkläre die Funktion + Skizze

MVC unterteilt eine interaktive Applikation in drei Teile: Model, View und Controller.

- Controller und View befinden sich im Presentation Layer und haben gegenseitig Abhängigkeiten
- Das Model darf keine Abhängigkeiten haben (Controller und View hängen vom Model ab)



4.1.1 Model

- Es befinden sich Teile im Domain und Data Source Layer.
- Das Model enthält die Kernfunktionalität und Daten. (z.B.: Datenbankzugriff)
- Im Projekt wird dies durch die Ordner *service* und *data* repräsentiert

4.1.2 View

- Im Projekt im Ordner *webapp* zu finden.
- Enthält im Projekt xhtml Dateien zur Darstellung und User Interaktion

4.1.3 Controller

- Im Projekt sind Controllerklassen im Ordner *web* zu finden.
- Sie enthalten die Logik und behandeln Benutzereingaben

5 Front Controller

5.1 Erkläre die Funktion + Skizze

- Client schickt Request an Front Controller
- FC erfasst nur Infos die er für die weiter Delegation braucht
- FC gibt Request an entsprechenden ConcreteCommand oder View weiter
- es gibt zwei Implementierungsvarianten des Controller
  - Servlet
  - ConcreteCommand

5.2 Servlet

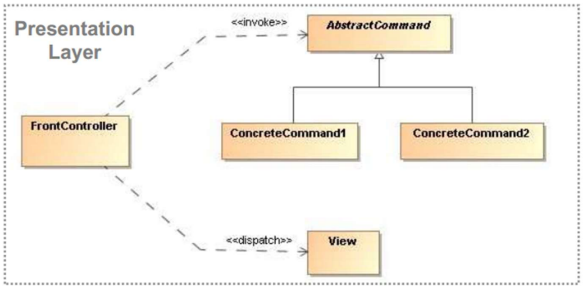
- Im Projekt wurde der Front Controller in Form eines Servlet realisiert,
- die Einbindung erfolgt in der Konfigurationsdatei *src/main/webapp/WEB-INF/web.xml*,
- Servlet ist eine Java-API, welche auf einem Server betrieben wird,

- die Verarbeitung von Requests und Responses wird ermöglicht,
- JSF und JSP können darauf aufsetzen, in unserem Projekt wurde JSF verwendet

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ↪  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  ↪  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  ↪  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
...
<servlet>
<servlet-name>Faces Servlet</servlet-name>
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>Faces Servlet</servlet-name>
<url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

5.2.1 Java Server Faces (JSF)

- JSF basiert auf dem MVC-Pattern
- JSF-View-Code ist im Projekt im Ordner *src/main/webapp/** zu finden
- JSF-Logik befindet sich in den Java-Beans (im Projekt */src/main/java/at/fhj/swd/psoe/web/**)
- in unserem Projekt gibt es zu jeder xhtml-View eine eigene Controller-Klasse, welche dem ViewHelper-Pattern entspricht
- in unserem Projekt kommt PrimeFaces zum Einsatz (eine konkrete Implementierungsart von JSF => Einbindung in pom.xml)




```
<!-- Pfad: /src/main/webapp/community/documentManagement.xhtml -->

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  ↪  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
```

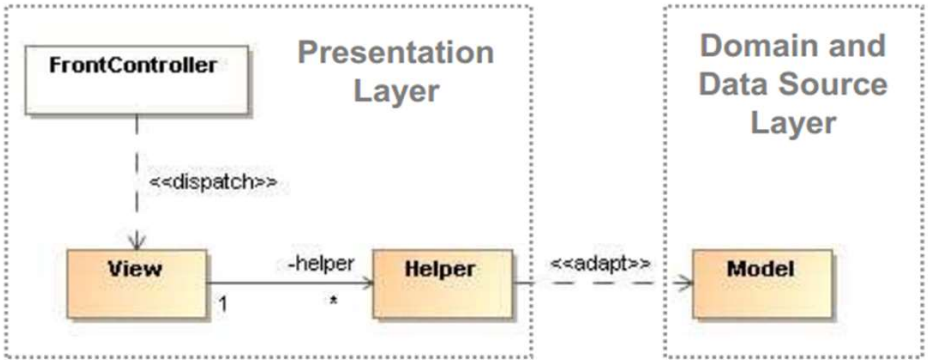
```
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:p="http://primefaces.org/ui"
xmlns:f="http://xmlns.jcp.org/jsf/core"
template="communityTemplate.xhtml">
<ui:define name="communityContent">
<h1>#{msg.document_manage_title}</h1>
<f:metadata>
<f:viewAction action="#{documentListController.loadDocumentsFromCommunity()}" />
</f:metadata>

<h:form id="doclistform">
<p:commandButton value="Refresh list"
↪  actionListener="#{documentListController.loadDocumentsFromCommunity()}"
↪  update="@form doclistform"></p:commandButton>
<p:dataTable id="doclisttable" value="#{documentListController.communityDocuments}"
↪  var="docs">
<p:column class="documenttimecolumn"
↪  headerText="#{msg.document_uploaded}">#{docs.createdTimestamp}</p:column>
<p:column class="documenttimecolumn"
↪  headerText="#{msg.label_userid}">#{docs.user.userId}</p:column>
<p:column headerText="#{msg.label_filename}">#{docs.filename}</p:column>
<p:column headerText="" class="documentbuttoncolumn">
<p:commandButton value="#{msg.button_download}" ajax="false"
onclick="PrimeFaces.monitorDownload(start, stop);">
<p:fileDownload value="#{documentController.downloadDocument(docs.id)}"/>
</p:commandButton>
</p:column>
<p:column headerText="" class="documentbuttoncolumn">
<p:commandButton id="btnDel" value="#{msg.button_delete}"
actionListener="#{documentController.removeDocument(docs.id)}"
update="@form doclistform">
</p:commandButton>
</p:column>
</p:dataTable>
</h:form>
<h:form id="formdocupload" enctype="multipart/form-data">
<p:fileUpload id="fileupload"
dragDropSupport="false"
update="@form doclistform"
fileUploadListener="#{documentController.uploadDocument}"
allowTypes="/(\.|\/)(pdf|jpe?g|docx)\$/" sizeLimit="5000000"
mode="advanced" label="Add document (.pdf .jpg .docx)">
</p:fileUpload>
</h:form>
<p:messages id="feedbackBox" severity="info,error" showDetail="true"
↪  showSummary="false">
<p:autoUpdate/>
</p:messages>
</ui:define>
</ui:composition>
```

5.3 Nenne die Konsequenzen der Anwendung

- es muss nur EIN (Front) Controller konfiguriert werden
- da bei jedem Request ein neues Command Objekt erzeugt wird ist Thread-Safety nicht notwendig
- da nur EIN Controller sind auch Erweiterungen durch z.B.: Decorator einfach (auch zur Laufzeit)

6 View Helper (*/src/main/java/at/fhj/swd/psoe/web/**)

6.1 Erkläre die Funktion + Skizze



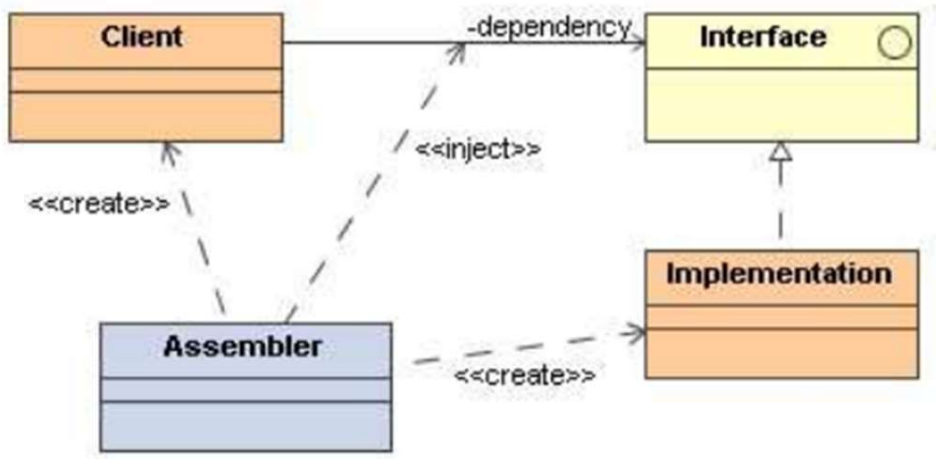
- View (xhtml-Dateien im Ordner */src/main/webapp/**) delegiert Aufgaben an Helper (z.B. DocumentController im Ordner web)
- Helper adaptieren View zu Model (Klassen in den Ordnern *src/main/java/at/fhj/swd/psoe/service/** und *src/main/java/at/fhj/swd/psoe/data/**)
- in View befindet sich HTML Code im ViewHelper Java Code zur Aufbereitung der Daten (+ wenig HTML)

6.2 Nenne die Konsequenzen der Anwendung

- kapselt Design-Code in View und View-Processing-Code Logik in Helper
- steigert Wiederverwendbarkeit, Wartbarkeit und Strukturierungsqualität der Anwendung
- vereinfacht Tests (Helperfunktionen ohne View)
- bessere Trennung zwischen
  - Presentation und Data Source Layer
  - Entwickler und Designer

7 Dependency Injection (CDI-Framework -> eingebunden im *./pom.xml*)

7.1 Erkläre die Funktion + Skizze



- ein Hauptgrund für Dependency Injection ist die Trennung zwischen Business- und Instanzierungslogik
- Klassen sollen nur noch Abhängigkeiten auf Interfaces haben
- die Verwendung des new Operators stattdessen würde eine Abhängigkeit zur Implementierungsklasse herstellen
- Implementierungsklassen sollen auf Interfaces referenzieren, wobei es ihnen egal wie diese Referenz hergestellt wird
- Die Instanziierung wird in die sogenannte Assembler Klasse "'herausgezogen"'
- Objekte werden mittels des Assembler "'gebaut"'.
- Zudem instanziiert er die Objekte und injected die Referenzen
- So hat er Client nur mehr eine Abhängigkeit auf das Interface
- wird die Implementierung getauscht, so geschieht das direkt im Assembler ohne den Client zu beeinflussen
- der Container erledigt DI über @Inject indem er prüft welche Klasse das Interface implementiert und entsprechende Referenz bei der Annotation einhängt
- Es wird zwischen Constructor Injection und Setter Injection unterschiedlichen

```
1         // Constructor Injection
2     puplic class Client {
3         private Interface iface;
4     public Client(Interface iface) {
```

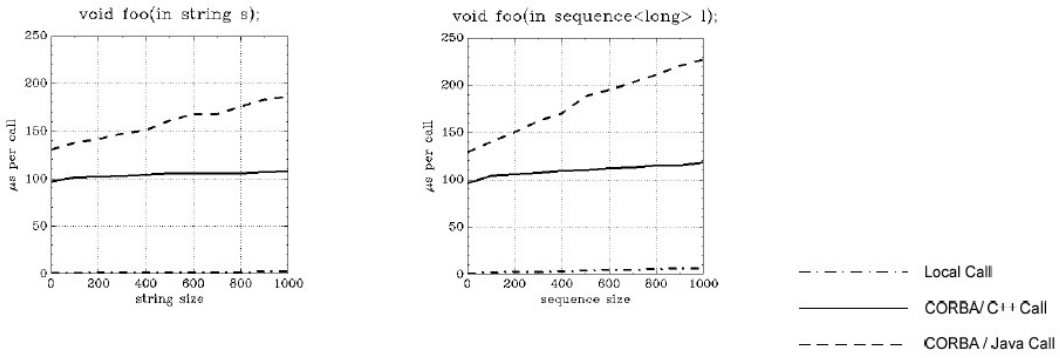
```
5           this.iface = iface;
6       }}
7
8           // Setter Injection
9       puplic class Client {
10          private Interface iface;
11      public setIface(Interface iface)
12      {
13          this.iface = iface;
14      }}
```

7.2 Nenne die Konsequenzen der Anwendung

- loose gekoppelte Objekte
- Referenzen nur noch auf Interfaces
- hohe Flexibilität (Strategy, Proxy,..)
- bessere Erweiterbarkeit und Testbarkeit

8 Data Transfer Object (DTO) Pattern

8.1 Erkläre die Funktion (Skizze - ein Grund für DTO)



- Transportiert Daten zwischen Prozessen um Remote Methodenaufrufe zu minimieren
- besteht aus Fields, Getter und Setter
- fasst Daten verschiedener Objekte zusammen die vom Remote Objekt benötigt werden
- ev. Map, Record Set, ... -> um Anzahl der Aufrufe zu minimieren

```
1  package at.fhj.swd.psoe.service.dto;
2
3  import java.io.Serializable;
4  import java.util.Date;
5
```

```java
public class DocumentDTO implements Serializable {
		private static final long serialVersionUID = 4016557982897997689L;

	private Long id;
	private Long documentlibraryID;
	private String filename;
	private UserDTO user;
	private byte[] data;
	private Date createdTimestamp;

	public DocumentDTO() {}

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public Long getDocumentlibraryID() {
		return documentlibraryID;
	}

	public void setDocumentlibraryID(Long documentlibraryID) {
		this.documentlibraryID = documentlibraryID;
	}

	public String getFilename() {
		return filename;
	}

	public void setFilename(String filename) {
		this.filename = filename;
	}

	public UserDTO getUser() {
		return user;
	}

	public void setUser(UserDTO user) {
		this.user = user;
	}

	public byte[] getData() {
		return data;
	}

	public void setData(byte[] data) {
		this.data = data;
	}

	public Date getCreatedTimestamp() {
```

```
59                  return createdTimestamp;
60          }
61
62          public void setCreatedTimestamp(Date createdTimestamp) {
63                  this.createdTimestamp = createdTimestamp;
64          }
65
66          @Override
67          public String toString() {
68                  return "DocumentDTO{" +
69                          "id=" + id +
70                          ", documentlibraryID=" + documentlibraryID +
71                          ", filename='" + filename + '\'' +
72                          '}';
73          }
74
75          @Override
76          public boolean equals(Object o) {
77                  if (this == o) return true;
78                  if (!(o instanceof DocumentDTO)) return false;
79
80                  DocumentDTO that = (DocumentDTO) o;
81
82                  return id.equals(that.id);
83          }
84
85          @Override
86          public int hashCode() {
87                  return id.hashCode();
88          }
89  }
```

8.2 Konsequenzen der Anwendung

- kapselt und versteckt
- nimmt Komplexität
- steigert Effizienz da weniger Aufrufe über Remotegrenze

Part II
Exception Handling

9 Checked und Runtime Exceptions in Java

9.1 Checked Exceptions (z.B. SQL-Exception)

- leiten von Exception Klasse ab und müssen behandelt werden (trows - catch)
- Verwendung für Probleme die durch User behoben werden können (alternative Aktion)

9.2 Unchecked Exceptions (z.B. NullPointerException)

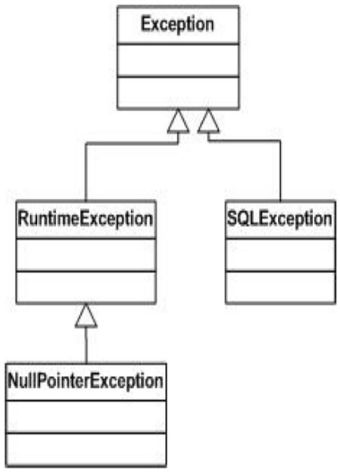
- leiten von RuntimeException ab
- Verwendung für technische Probleme (User kann nichts machen außer neu starten)
  - beschädigte Datenbank - die Exception geht durch alle Layer
    - erst mit Implementierungsspezifischer Exception
    - später mit Runtime **ohne Stacktrace** bis zum User (-> Destructive wrapping mit Log and Throw im ServiceLayer)
    - im Projekt: ServiceException, DaoException, AuthenticationException und SerializerException

```
1  package at.fhj.swd.psoe.service;
2
3  public class ServiceException extends RuntimeException {
4      private static final long serialVersionUID = -1109707847007116930L;
5
6      public ServiceException(String message) {super(message);}}
```

```
1      package at.fhj.swd.psoe.data;
2
3      public class DaoException extends RuntimeException {
4          private static final long serialVersionUID = -2712863481296295032L;
5
6          public DaoException(String message, Throwable cause) {
7              super(message, cause);
8      }
9      public DaoException(Throwable cause) {super(cause);}}
```

10 Best Practice Beispiele beim Einsatz von Exceptions

- Exceptions nicht für Programmflusskontrolle verwenden (schlechte Performance)
- offene Ressourcen schließen (try-with-resources bzw. close im finally)
- selbst erstellte Exceptions auch mit nützlichen Infos ausstatten



- Implementierungsspezifische Exceptions nicht bis zum User durchwerfen (stattdessen catch + trow RuntimeException)
- dokumentieren mit @trows im DOC, testen mit JUnit

11 Exception Handling Anti Pattern

- Log and Trow (nie beides: entweder, oder)
- Trowing Exception bzw. catch Exception (spezifischere anstatt Basisklasse verwenden)
- Destructive Wrapping (wenn bei catch + trow = wrapping nicht die Original Exception weitergegeben wird)
- Log and return Null (provoziert an einer anderen Stelle eine NullPointerException)
- Catch and Ignore
- Unsupported Operation return Null (besser UnsupportedOperationException)

12 Destructive Wrapping im Service Layer

- im Codebeispiel wird in Zeile 12 eine IllegalArgumentException (Runtime) gefangen in Zeile 18 die Exception allgemein
- beide werden in Zeile 14 bzw. 18 inklusive Stacktrace geloggt
- ausnahmsweise muss hier zusätzlich auch eine neue ServiceException geschmissen werden, jedoch **ohne Stacktrace** (siehe Zeile 15 und 19)

```
1          package at.fhj.swd.psoe.service.impl;
2          ...
3          @Override
4          public List<DocumentDTO> getDocumentsFromCommunity(Long communityID) {
5              try {
```

```
6                   List<Document> documents;
7                   if (communityID <= 0) throw new IllegalArgumentException("community
                    ↪  must not be empty");
8                   Community community = communityDAO.findById(communityID);
9                   if (community == null) throw new IllegalStateException("community "
                    ↪  + communityID + " not found");
10                  documents = documentDAO.findByCommunity(community);
11                  return
                    ↪  documents.stream().map(DocumentMapper::toDTO).collect(Collectors.toList());
12                  } catch (IllegalArgumentException iaex) {
13                      String errorMsg = "Could not load docs from community (illegal
                        ↪  argument)";
14                      logger.error(errorMsg, iaex);
15                      throw new ServiceException(errorMsg);
16                      } catch (Exception ex) {
17                          String errorMsg = "Could not load docs for community.";
18                          logger.error(errorMsg + " id " + communityID, ex);
19                          throw new ServiceException(errorMsg);
20                          }
21                          }
22                          ...
```

13 Exception Testing

```
1                           package at.fhj.swd.psoe.service;
2
3                           import java.util.ArrayList;
4                           ...
5
6                           @RunWith(MockitoJUnitRunner.Silent.class)
7                           public class DocumentServiceTest {
8                               @Mock
9                               private DocumentDAO documentDAO;
10                              ...
11
12                              @Test(expected = ServiceException.class)
13  public void getDocumentsFromCommunity_WithId0_ShouldFail() {
14      documentService.getDocumentsFromCommunity(0L);
15      }
16
17      @Test(expected = ServiceException.class)
18      public void getDocumentsFromCommunity_NoDocuments_ShouldFail() {
19          Community community = Mockito.mock(Community.class);
20          Mockito.when(community.getId()).thenReturn(COMMUNITYID);
21          Documentlibrary doclib = Mockito.mock(Documentlibrary.class);
22          Mockito.when(community.getDocumentlibrary()).thenReturn(doclib);
23          Mockito.when(doclib.getCommunity()).thenReturn(community);
24          Mockito.when(communityDAO.findById(community.getId())).thenReturn(null);
25
26      documentService.getDocumentsFromCommunity(COMMUNITYID);
27      }
28      ...
```

Part III
Allgemeines & Config

14 Logging

14.0.1 Vorteile Logging mittels Framework (z.B.: log4j)

- Nutzt ein einheitliches Format / Konventionen
- logging kann optional an und ausgeschalten werden
- durch verschiedene Log-level können Logs gefiltert erstellt werden
- Layout für Ausgabe kann zentral definiert/geändert werden

15 Annotationen

- @MappedSuperclass
  - ist im Hybernate Framework
  - eine Klasse durch die gemeinsame Felder definiert werden.
  - definiert eine abstrakte Superklasse
- @Produces
  - kommt während deployment, markiert Factory Method damit man nicht direkt auf die Klasse zugreifen muss
- @Typed
  - zeigt die Vererbung Wieso bei uns allein stehend?
- @Named
  - Zeigt bei Mehrdeutigkeit das richtige Objekt mit dem Namen
- @Resource
  - fast wie Dependency Injection
  - damit weiß der Container wie er das Annotierte Feld instanzieren muss
- @Stateless
  - speichert den Client Status nicht
- @Entity
  - Data Access Layer
- @Table

  - Tabellenname im SQL
- @Column
  - SQL-Spalten nullable=false
- @OneToMany
- @JoinColums
  - welche Spalten zusammen gehören FK
- @OneToMany
  - auf anderen Seite
- @ApplicationScoped
  - lebt die ganze Applikation lang, wird einmal gemacht.
- @PersistenceContext
  - persistance.xml auslesen für Treiber und andere JPA Geschichten + Data Source. Entity Manager
- @Id
  - das ist die id
- @GeneratedValue
  - Wert kommt aus der DB
- @Local
  - Klasse für lokale Aufrufe.
- @Remote
  - interprozessaufrufe. RMI
- @ApplicationException
  - Rollback wenn so eine Exception kommt, Nachricht zum Client.

15.1 Annotationen - Details

- CascadeType.Remove löscht die damit Annotierte Verknüpfung mit
- dies geht auch rekursiv in der kompletten Datenbank
- CascadeType.Remove und orphanRemoval=true ist equivalent

```
    ...
    @OneToOne(fetch = FetchType.EAGER,
            cascade = CascadeType.ALL,
            orphanRemoval = true)
    @JoinColumn(name = "documentlibrary_id")
    private Documentlibrary documentlibrary;

    @Column(nullable = false, unique = true)
    private String name;
    ...
    public enum CascadeType {
        /* Cascade all operations /
        ALL,
        /* Cascade persist operation /
        PERSIST,
        /* Cascade merge operation /
        MERGE,
        /* Cascade remove operation /
        REMOVE,
        /* Cascade refresh operation /
        REFRESH,
         */
        DETACH
        ...
    }
)
```

16 Konfigurationsdateien

16.1 persistence.xml

Die Datei *persistence.xml* ist der zentrale Bestandteil der Persistierungs-Konfiguration. Folgende Dinge können konfiguriert werden:

- SQL dialect
- the persistence provider that shall be used at runtime
- the data source you want to use to connect to your database
- several provider-specific configuration parameters

Listing 1: persistence.xml

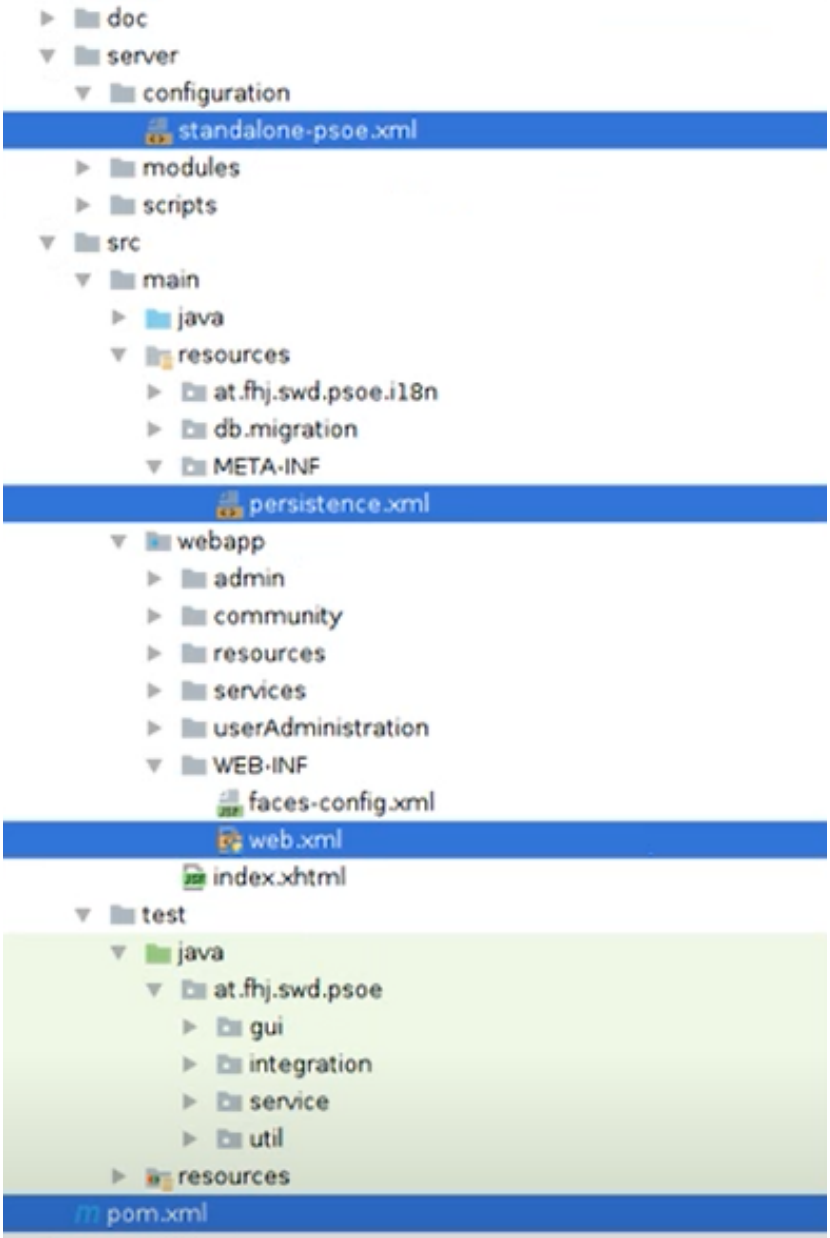
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
xmlns="http://java.sun.com/xml/ns/persistence"
↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
↪  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

<persistence-unit name="primary">
```

```
<jta-data-source>java:jboss/datasources/psoeDS</jta-data-source>
<properties>
<!-- Properties for Hibernate -->
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"
↪  />
<property name="hibernate.enable_lazy_load_no_trans" value="true" />
<!--
SQL stdout logging
-->
<property name="hibernate.show_sql" value="true"/>
<property name="hibernate.format_sql" value="false"/>
<property name="use_sql_comments" value="true"/>

<!--
<property name="hibernate.hbm2ddl.auto" value="create-drop" />
-->
</properties>
</persistence-unit>

</persistence>
```

16.2 web.xml

- konfiguriert den Java Webserver (Wildfly - JBOSS)
- Einbindung des Faces-Servlet (FrontController - Implementierung, Zugriffskontrolle, Rollenkonfiguration)
- befindet sich im Ordner **src/main/webapp/WEB-INF/web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
↪  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
↪  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
↪  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">

<servlet>
<servlet-name>Faces Servlet</servlet-name>
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>Faces Servlet</servlet-name>
<url-pattern>*.xhtml</url-pattern>
</servlet-mapping>

<!-- Security roles -->
<security-role>
<description>administrators</description>
<role-name>ADMIN</role-name>
</security-role>

```


doc
server
configuration
standalone-psoe.xml
modules
scripts
src
main
java
resources
at.fhj.swd.psoe.i18n
db.migration
META-INF
persistence.xml
webapp
admin
community
resources
services
userAdministration
WEB-INF
faces-config.xml
web.xml
index.xhtml
test
java
at.fhj.swd.psoe
gui
integration
service
util
resources
pom.xml

```
<!-- Security constraints -->
<security-constraint>
<web-resource-collection>
<web-resource-name>admin area</web-resource-name>
<url-pattern>/admin/*</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>ADMIN</role-name>
</auth-constraint>
</security-constraint>

<login-config>
<auth-method>FORM</auth-method>
<realm-name>pse</realm-name>
<form-login-config>
<form-login-page>/login.xhtml</form-login-page>
<form-error-page>/login.xhtml</form-error-page>
<!--                    <form-error-page>/loginerror.xhtml</form-error-page> -->
</form-login-config>
</login-config>
</web-app>
```

16.3 pom.xml

- *resources*-plugin (bindet die Serverressourcen ein - Ordner *configuration* im Projekt - z.B. *standalone-psoe.xml*)
- Wildfly (JBoss)– Webserver

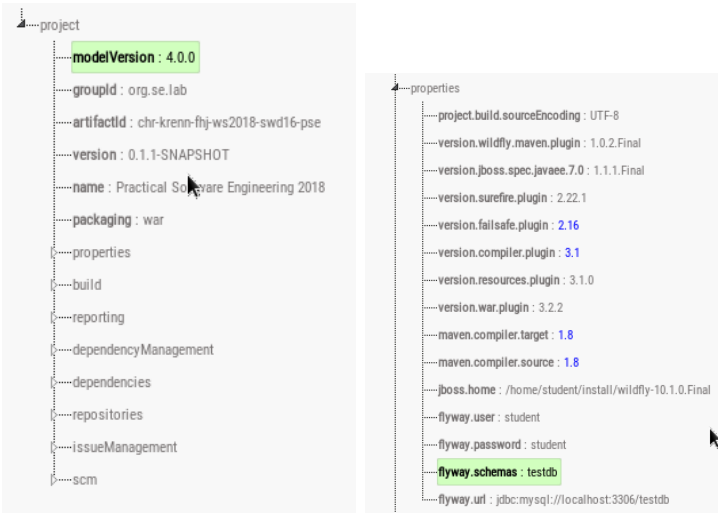
1. Compile
2. Surefire (unitTests)
3. Packaging - war file erstellen
4. Wildfly - fressen und deployen
5. Failsafe IT-test
6. MVN site
7. Gui test

- Primeface = jsf Framework
- Jacoco = test Coverage
- Slf4j = logger
- Jaxb – xml
- Cdi = context dependancy injection

16.3.1 Aufbau pom.xml

16.4 standalone-psoe.xml

Wird ein JBoss Applikationsserver im *standalone*-Modus betrieben, läuft jede Instanz in einem eigenen Prozess. Diese Datei ist eine Java Enterprise Edition 6 zertifizierte Web-Profil Konfiguration welche alle benötigten Technologien (z.B. Extensions von JBoss, Datasources etc.) definiert.



JBoss EAP benutzt standardmäßig die standalone.xml Konfigurationsdatei, kann aber auch unter Verwendung einer anderen gestartet werden.

Abschnitte der standalone.xml

- extensions (z.B. diverse Wildfly Module)
- management (z.B. Access Control -> role-mapping)
- profile (z.B. JPA Subsystem)
- interfaces (z.B. $jboss.bind.address:127.0.0.1)
- socket-binding-group (z.B $jboss.http.port:8080)
- Rechte (Management-Realm)
- Datenbankzugriffsparameter

Listing 2: standalone.xml (auszugsweise)

```
1  ........
2
3              </endpoint-config>
4  <client-config name="Standard-Client-Config"/>
5  </subsystem>
6  <subsystem xmlns="urn:jboss:domain:weld:3.0"/>
7  </profile>
8  <interfaces>
9  <interface name="management">
10 <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
11 </interface>
12 <interface name="public">
13 <inet-address value="${jboss.bind.address:127.0.0.1}"/>
14 </interface>
15 </interfaces>
```

```
<socket-binding-group name="standard-sockets" default-interface="public"
↪ port-offset="${jboss.socket.binding.port-offset:0}">
<socket-binding name="management-http" interface="management"
↪ port="${jboss.management.http.port:9990}"/>
<socket-binding name="management-https" interface="management"
↪ port="${jboss.management.https.port:9993}"/>
<socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
<socket-binding name="http" port="${jboss.http.port:8080}"/>
<socket-binding name="https" port="${jboss.https.port:8443}"/>
<socket-binding name="txn-recovery-environment" port="4712"/>
<socket-binding name="txn-status-manager" port="4713"/>
<outbound-socket-binding name="mail-smtp">
<remote-destination host="localhost" port="25"/>
</outbound-socket-binding>
</socket-binding-group>
</server>
```

16.5 log4j.properties

textitsrc/test/resources/log4j.properties

17 Fehler im Projekt

17.1 Return null

Anstatt von Null einfach eine Leere Liste bzw. ein default Objekt (oder new <Object>) zurückgeben

17.2 Exception nicht gefangen

```
    package at.fhj.swd.psoe.service.impl;
    ...
    @Override
    public void removeCommunityByAdmin(CommunityDTO communityDTO) {
        Community community = communityDAO.findById(communityDTO.getId());
        String errorText;
        try {
            communityDAO.removeCommunityByAdmin(community);
        } catch (DaoException e) {
            errorText = "Error removing community";
            logger.error(errorText, e);
            throw new ServiceException(errorText);
        }
    }
```

17.3 Destructive Wrapping - Logging fehlt - Information geht verloren

```
    package at.fhj.swd.psoe.service.impl;
    ...
@Override
public CommunityDTO updateCommunityEnabled(String adminUserId, String communityName,
↪ boolean isEnabled) {
```

```
    String errorText = "";
    try {
        boolean hasPermission = false;
        User adminUser = userDao.getByUserId(adminUserId);
        Community community = communityDAO.getByName(communityName);


        for (Role r : adminUser.getRoles()) {
            if (r.getname().equals("ADMIN") || r.getname().equals("PORTALADMIN")) {
                hasPermission = true;
            }
        }
        if (hasPermission ||
        ↪  adminUser.getUserId().equals(community.getCommunityAdminUser().getUserId()))
        ↪  {
            community.setIsEnabled(isEnabled);
            communityDAO.update(community);
        } else {
            errorText = "No Permission to update community";
            throw new AuthenticationException(errorText);
        }
        return CommunityMapper.toDTO(community);
    } catch (DaoException e) {
        errorText = "Error updating community enabled";
        throw new ServiceException(errorText);
    } catch (AuthenticationException e) {
        throw new ServiceException(errorText);
    } catch (Throwable e) {
        errorText = "Unknown error updating community enabled";
        throw new ServiceException(errorText);
    }
}
```

17.4 Logger mit falschem Parameter

```
    package at.fhj.swd.psoe.service.impl;
...
    @Stateless
    public class DepartmentHierarchyServiceImpl implements
    ↪  DepartmentHierarchyService, Serializable {

        private static final long serialVersionUID = -2467949382018996094L;
        private static final Logger logger =
        ↪  LoggerFactory.getLogger(UserServiceImpl.class);
...
```

17.5 Fehlendes Exception Handling

```
package at.fhj.swd.psoe.service.impl;
...
@Local(MessageService.class)
@Remote(MessageServiceRemote.class)
@Stateless
```

```
public class MessageServiceImpl implements MessageService, MessageServiceRemote,
↪  Serializable {
    ...
@Override
    public MessageDTO getByMessageId(long id) {
        Message message = messageDAO.getById(id);
        if (message == null) {
            return null;
        }
        return MessageMapper.toDTO(message);
    }

    @Override
    public MessageDTO updateByAdmin(long id, String content, Date changed) {
        Message message = messageDAO.getById(id);
        message.setContent(content);
        message.setEditedByAdmin(changed);
        return MessageMapper.toDTO(messageDAO.update(message));
    }

    @Override
    public MessageDTO updateByUser(long messageId, String content, Date changed) {
        Message message = messageDAO.getById(messageId);
        message.setContent(content);
        message.setEditedByUser(changed);
        return MessageMapper.toDTO(messageDAO.update(message));
    }
```

17.6 Exception werfen und gleich wieder fangen

```
    ...
public class MessageServiceImpl implements MessageService, MessageServiceRemote,
↪  Serializable {
    private static final long serialVersionUID = 6768291437557855130L;
    ...
// nicht optimal, da die IllegalArgumentException gleich wieder gefangen wird
// überdies wird alles andere nicht gefangen


    @Override
    public void deleteMessage(long id) {
        try {
            Message message = messageDAO.getById(id);
            if (message == null) {
                throw new IllegalArgumentException("Message cannot be empty");
            }
            messageDAO.delete(message);
            logger.info("Message deleted successfully");
        }
        catch (IllegalArgumentException ex) {
            String errorMsg = "Could not delete the message (illegal argument)";
            logger.error(errorMsg, ex);
            throw new ServiceException(errorMsg);
```

```
        }
    }

    //---------- besser wäre
    package at.fhj.swd.psoe.service.impl;
...
// erst loggen, dass man in die Methode gekommen ist
// wenn userDTO null ist wird IllegalArgument geworfen und das außerhalb des try
↪  catch blocks
// erst wird die DaoException gefangen und anschließend alle anderen
// Stacktrace wird geloggt und jeweils die ServiceException weitergegeben
@Override
public void saveUser(UserDTO userDTO) {
    logger.debug("UserService saveUser() called with parameter: '{}'", userDTO);

    if (userDTO == null) {
        throw new IllegalArgumentException("userDTO is null");
    }

    try {
        User user = (userDTO.getId() == null) ? new User() :
        ↪  userDao.getById(userDTO.getId());
        userDao.update(UserMapper.toEntity(userDTO, user));
    } catch (DaoException e) {
        logger.error("Error saving user", e);
        throw new ServiceException("Error saving user");
    } catch (Throwable e) {
        logger.error("Unknown error saving user", e);
        throw new ServiceException("Unknown error saving user");
    }
}
```

18 Tests

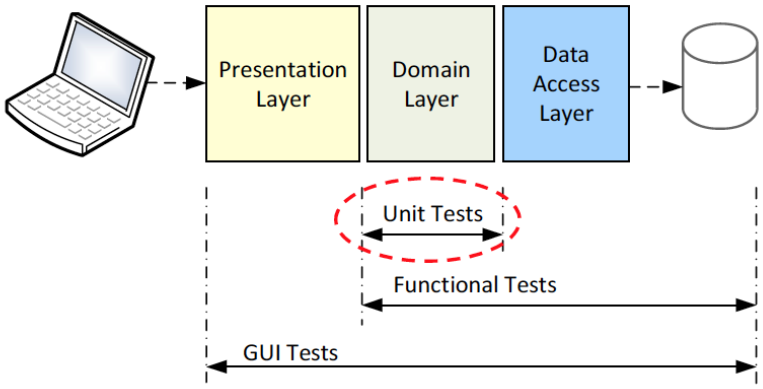


Figure 1

- Ziel der Tests ist es, 100% Codecoverage im Service-Layer zu erreichen (Anforderung für unser Projekt),
- Private und Protected-Methoden müssen per se nicht getestet werden (Rule of Thumb),Wege zum erfolgreichen Test:
  - Access-Modifier der Methoden auf public (public static) ändern,
  - Extrahieren der Methoden in neue Klassen,
  - Notfalls Package-Sichtbarkeit in Methode, wenn diese privat bleiben muss, Tests aber unbedingt notwendig sind.

18.1 Testpyramide

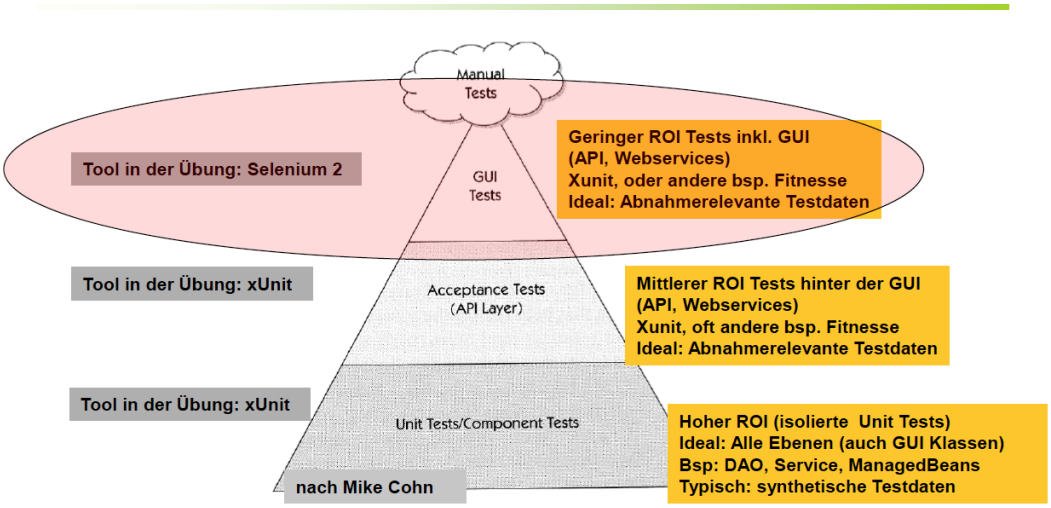


Figure 2

18.2 Unit

- Im Build-Cycle werden zuerst Unit-Tests ausgeführt,
- sie befinden sich im Projekt im Ordner *src/test/at/fhj/swd/psoe/service/*Test.java*,
- ein Unit-Test muss jede Methode der Service-Klasse testen
- Mockito imitiert die Methoden, die von der Datenbank abhängen, da diese zu diesem Zeitpunkt noch nicht zur Verfügung steht (Build-Cycle => Compile - Unit-Tests)
- Im Unit-Test wird jeweils die kleinste Einheit getestet
- Exceptions einfach mit Mockito werfen und testen, ob sie geworfen wurden

```java
@Test(expected = ServiceException.class)
public void testGetCommunityByUserMessagesDAOException()
{
Mockito.when(this.userPrincipal.getId()).thenReturn(-1L);
Mockito.when(this.communityService.loadAllCommunitiesByUser(-1L)).thenThrow(new
↪  DaoException(new RuntimeException()));

this.activitystreamService.getCommunityByUserMessages();
}
```

```java
@Test
public void getDocumentsFromCommunity_ShouldReturnDocuments() {
        Community community = Mockito.mock(Community.class);
        Mockito.when(community.getId()).thenReturn(DOCUMENTID);
        Documentlibrary doclib = Mockito.mock(Documentlibrary.class);

        Mockito.when(community.getDocumentlibrary()).thenReturn(doclib);
        Mockito.when(doclib.getCommunity()).thenReturn(community);
        User user = Mockito.mock(User.class);

        Mockito.when(communityDAO.findById(community.getId())).thenReturn(community);

        List<Document> documents = prepareTwoDocuments(doclib,user);
        Mockito.when(documentDAO.findByCommunity(community)).thenReturn(documents);

        List<DocumentDTO> documentsDTO =
        ↪  documentService.getDocumentsFromCommunity(community.getId());

        Assertions.assertThat(documentsDTO)
        .usingElementComparatorIgnoringFields("user")
        .containsExactlyInAnyOrder(DocumentMapper.toDTO(documents.get(0)),
        ↪  DocumentMapper.toDTO(documents.get(1)));
}
```

18.3 Integration Tests

- testen alle Komponenten mit (Datenbank)
- werden im Build-Cycle erst nach dem Deployen des WAR-Files ausgeführt (laufende Applikation)
- In unserem Projekt kommt das Plugin *failsafe* zum Einsatz
- Datenbankscripts werden separat zu den anderen Scripts ausgeführt (Datenbank vorbereiten und auf den alten Stand zurückbringen)
-

18.4 GUI-Test

18.4.1 Page Object Pattern

- stellt Screens der Web-App als Reihe von Objekten dar

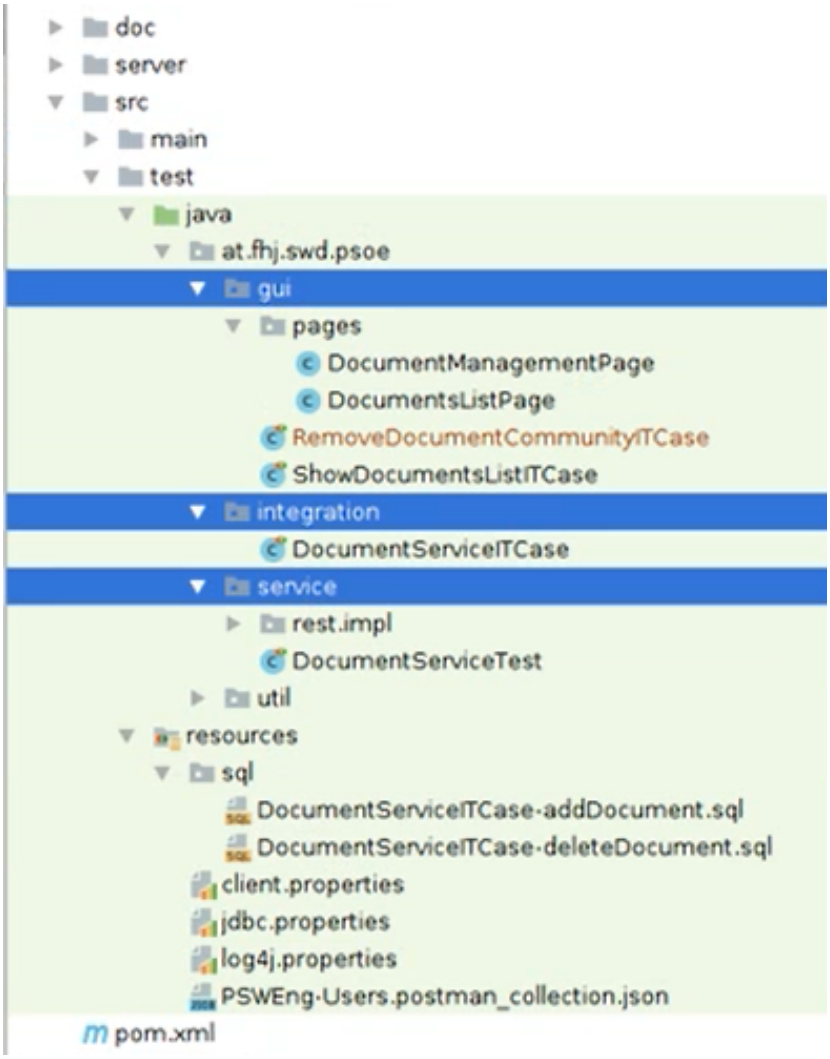


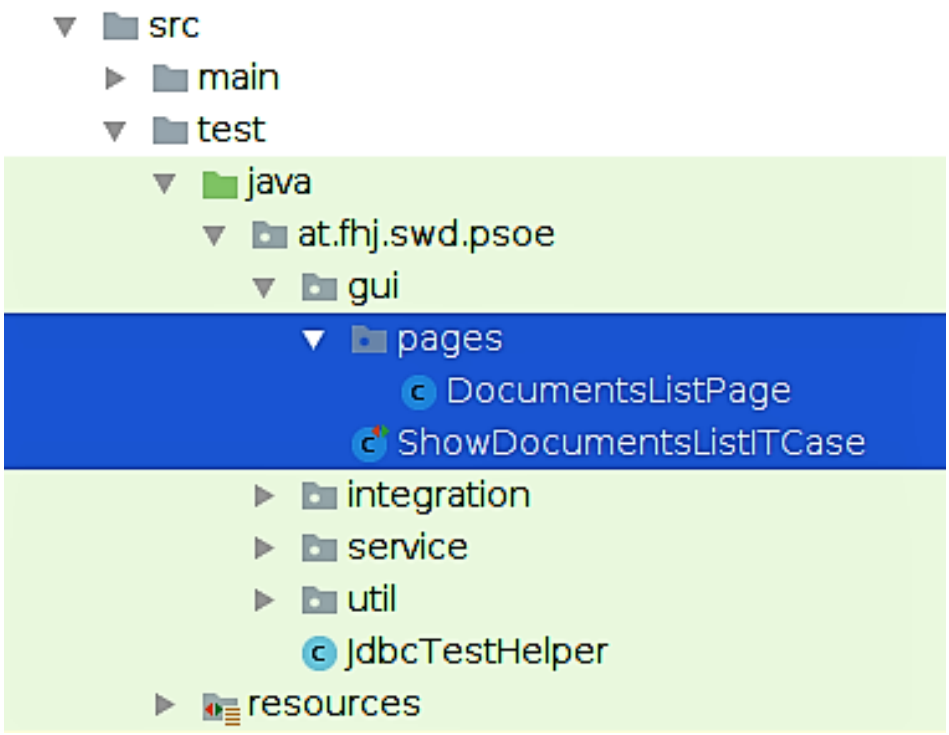
Figure 3

- tatsächlich werden nicht alle Seiten sondern wesentliche Elemente in Objekte gekapselt
- eine HTML Seite wird so mitunter mit mehreren Objekten dargestellt (z.B. Header und Footer Page Object)
- Das Page Objekt Design eignet sich besonders gut um Selenium Tests umzusetzen
- Mittels der Page Objekte kann HTML Code verändert werden (Verkapselung)
- ermöglichen die Modellierung der Benutzeroberfläche für Tests
- reduziert Code duplication
- verbessert Testwartbarkeit und macht Tests robuster

18.5 Page Object Pattern lt. Zahnlücke

- Trennung zwischen Testmethode und Page Code

- Je Page eine Klasse mit Services / Operationen
- Return einer Operation ist ein PageObject
- Einfachere Wartbarkeit (Kapselung in PageObject)



18.6 Beispiel aus dem Projekt

18.6.1 Integration GUI Test mit Selenium

```
1  package at.fhj.swd.psoe.gui.pages;
2
3  import org.openqa.selenium.By;
4  import org.openqa.selenium.WebDriver;
5  import org.openqa.selenium.WebElement;
6  import java.util.List;
7
8  public class DocumentsListPage extends AbstractPage {
9  private List<WebElement> list;
10
11 public DocumentsListPage(WebDriver driver) {
12 super(driver);
13 }
14 public List<WebElement> getList() {
15 list =
   ↪  this.getDriver().findElements(By.xpath("//*[@id=\"documents:comdoctable_data\"]"));
16 return list;
```

```
}

}
```

18.6.2 Durch Selenium getestetes Page Objekt

```
package at.fhj.swd.psoe.gui;

import at.fhj.swd.psoe.JdbcTestHelper;
import at.fhj.swd.psoe.gui.pages.*;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;
import java.sql.SQLException;
import java.util.stream.Collectors;

public class ShowDocumentsListITCase extends AbstractChromeTest {
final private String roleName = "ADMIN";
private LoginPage loginPage;
private WelcomePage welcomePage;
private DocumentsListPage documentsListPage;
private String baseUrl = "http://localhost:8080/chr-krenn-fhj-ws2018-swd16-pse";
private static final JdbcTestHelper JDBC_HELPER = new JdbcTestHelper();

@Before
@Override
public void setUp() {
super.setUp();
JDBC_HELPER.executeSqlScript(
"src/test/resources/sql/DocumentServiceITCase-addDocument.sql");
loginPage = new LoginPage(this.driver, baseUrl, 60);
welcomePage = loginPage.login("testdocument@swd.com", "admin");
CommunitiesPage communitiesPage = welcomePage.openCommunitiesPage();
CommunityPage communityPage = communitiesPage.openCommunityPage();
documentsListPage = communityPage.openDocumentListPage();
}

@Test
public void testTwoDocumentsListed()  {
String content = documentsListPage.getList().stream().map(x ->
↪  x.getText()).collect(Collectors.joining());
Assert.assertTrue(content.contains("documentuser123"));
Assert.assertTrue(content.contains("DocumentITCase1"));
Assert.assertTrue(content.contains("DocumentITCase2"));
}

@After
@Override
public void tearDown() {
super.tearDown();
```

```
47 JDBC_HELPER.executeSqlScript(
48 "src/test/resources/sql/DocumentServiceITCase-deleteDocument.sql");
49 }
50 }
```

19 Toni FRAAGNAA

Den Code durchgehen - was statt null - könnte leere Liste sein, return null sollte aber auch okay sein welche Exception - logger ok? ob ein Throw im try Block ok ist - sollte okay sein, da wenn nicht im try-Block, erfolgt kein Mapping als DAO-Exception.

```
1      @Override
2      public List<Document> findByUser(User user) {
3          logger.debug("dao: documents findByUser");
4          try {
5              if(user == null) throw new IllegalArgumentException("user must not be
               ↪  null");
6              return entityMangaer.createQuery("from Document d where d.user.userId =
               ↪  :userid", Document.class).setParameter("userid",
               ↪  user.getUserId()).getResultList();
7          } catch(NoResultException noresex) {
8              logger.info("dao: findByUser no documents found");
9              return null;
10         }
11         catch (Exception e) {
12             throw new DaoException("error finding documents of user",e);
13         }
14     }
15
16     // throw in try - passt
17         @Override
18     public void delete(Document document) {
19         logger.debug("dao: delete document");
20
21         try {
22             if(document == null) throw new IllegalArgumentException("document must
               ↪  not be null");
23             if(!entityMangaer.contains(document)) {
24                 document = entityMangaer.merge(document);
25             }
26             entityMangaer.remove(document);
27         }
28         catch (Exception e) {
29             throw new DaoException("error deleting document",e);
30         }
31     }
32     // ist loggen ohne stacktrace ok? - Stack-Trace gehört eigentlich dazu
33        @Override
34     public TimeRecording getTimeRecordingByTask(Task task) {
35         TimeRecording result = new TimeRecording();
36         try {
```

```
37              result = entityManager.createQuery("select t from TimeRecording t where
                ↪  t.task.id = :id", TimeRecording.class)
38                      .setParameter("id", task.getId()).getSingleResult();
39          } catch (NoResultException e) {
40              logger.warn("Not Possible to find TimeRecording with task id " +
                ↪  task.getId());
41          }
42          return result;
43      }
44      // UserDTO - @XmlElement
45      @XmlElement - sichtbar in XmlSerializerTest - für den automatischen Import der
        ↪  User via XML
46      public void setPhoneNumber(String phoneNumber) {
47          this.phoneNumber = phoneNumber;
48      }
49      // wos is produces
50      @Override
51      @Produces - ist eine Factory-Methode - kommt während dem Deployment und erstellt
        ↪  das Objekt
52      @Named("userPrincipal")
53      @SessionScoped
54      public UserPrincipal getUserPrincipal() {
55          String principalName = sessionContext.getCallerPrincipal().getName();
56          logger.info(principalName);
57          try {
58              User user = userDao.getByEmail(principalName);
59              return new UserPrincipal(user.getId(), user.getUserId(),
                ↪  user.getEmail(), user.getFirstname(), user.getLastname());
60          } catch (DaoException e) {
61              logger.error("Error loading user '{}'", principalName, e);
62              throw new ServiceException("Error loading user");
63          } catch (Throwable e) {
64              logger.error("Unknown error loading user '{}'", principalName, e);
65              throw new ServiceException("Unknown error loading user");
66          }
67      // @Typed - zeigt bei Mehrdeutigkeit die Vererbung (z.B. implementiert Interface
        ↪  und leitet von Klasse ab - muss aber in Klammer immer mitgegeben werden,
        ↪  wovon Java Server Beans dann die Ableitung macht)
68
69      // warum immer mappedBy Mehrzahl
70      @ManyToMany(mappedBy = "businessTrips") - Name vom Feld in der Entity, die
        ↪  verbunden wird, da ManyToMany, ist es egal, bei welcher Tabelle
71
72      // dependency Injection - wird schon im Skript erklärt
73
74      // braucht man im Controller (ViewHelper) überhaupt noch Exception Handling - es
        ↪  passiert nur Nutzereingabenvalidierung und die Fehlermeldung kann nicht
        ↪  weitergeworfen werden - es wird nur mehr in den Logger geschrieben
75
76      //müssen wir die Folien genau beherrschen (Stubs vs. Mocks?) Nein
77
78      //Bei welchem Goal wird was mitausgeführt? IT-Test bei mvn wildfy:run?
79
```

```
80      //Woher weiß PrimeFaces, wie es zum Ordner web mit den Controllern kommt?
81
82      //MessagePrincipal - @Typed - bereits oben erklärt
83
84          //Wie löst Maven Abhängigkeiten zu Libraries auf? Es wirdim Maven-Repository
            ↪  nach dem Package gesucht und dort nach der Version, die in der
            ↪  Dependency angegeben wurde. Wird die Library nicht gefunden, muss ein
            ↪  alternatives Repo angegeben werden. WICHTIG: niemals Libraries mit
            ↪  Plugins verwechseln (Plugin ist eine Erweiterung der
            ↪  Maven-Funktionalität, Library ist ein bestehender Java-Code, der
            ↪  verwendet werden kann)
```

20 Frageart Prüfung

Welche Fehler können bei Exception-Handling vorkommen in unserem Projekt?? – wie funktioniert es grundsätzlich in unserem Code

DocumentDAO – DocumentService – DocumentController – so sollte Exception-Handling implementiert warden

DAO wirft Exception – im ServiceLayer wird dies gefangen und der Stack-Trace wird im weggeloggt und eine benutzerfreundliche Fehlermeldung wird ausgegeben (Destructive Wrapping).

Alle Patterns, die vorkommen – praktische Beispiele aus dem Code

Was sind JavaBeans? Wie funktioniert das Konzept? Wie wird es genau implementiert? NamedBean, TypedBean etc.