Practical Software Engineering

SWD14 WS 2016

Name: _____ **Datum: 10.2.2017**

Pro Frage können max. 8 Punkte erreicht werden.
Daraus ergibt sich eine max. Gesamtpunkteanzahl von 40 Punkten.

Punkte: _____

1) Erklären Sie den gegebenen Source Code:

   a. Beschreiben Sie das **Architectural Pattern**, welches hier verwendet wird (inkl. **Skizze**).

   b. **Welcher Teil** dieses Architectural Patterns wird vom vorliegenden Source Code implementiert?

   c. **Beschreiben Sie die Elemente** des gegebenen Source Codes im Kontext dieses Architectural Patterns.

```
<ui:composition
        xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://xmlns.jcp.org/jsf/html"
        xmlns:p="http://primefaces.org/ui"
        xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <h:form id="newsForm" rendered="#{facesContext.externalContext.isUserInRole('admin')}">
        <p:panelGrid columns="2">
            <h:outputText value="Titel:"/>
            <h:inputText value="#{newsBean.news.title}" id="newsTitleField"/>

            <h:outputText value="Inhalt:"/>
            <h:inputTextarea value="#{newsBean.news.message}" id="newsContentField"/>

            <h:outputText value="Gültig von:"/>
            <p:calendar id="activation" value="#{newsBean.activationDate}" mindate="#{newsBean.currentDate}"
                        pattern="dd.MM.yy"/>

            <h:outputText value="Gültig bis:"/>
            <p:calendar id="termination" value="#{newsBean.terminationDate}" mindate="#{newsBean.activationDate}"
                        pattern="dd.MM.yy"/>
        </p:panelGrid>

        <p:commandButton value="#{newsBean.buttonNewsText}" action="#{newsBean.addNews()}" id="addNews"/>
    </h:form>
</ui:composition>
```

2) Erklären Sie das gegebene **pom.xml** File:

   a. Was bedeuten die Elemente **<plugin>** und **<parent>**

   b. Erklären Sie, wie Maven **Abhängigkeiten zu Libraries** auflöst.

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>frontend</artifactId>
        <groupId>at.fhj.swd14.pse</groupId>
        <version>1.2-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>frontend-impl</artifactId>
    <packaging>war</packaging>

    <dependencies>
        <dependency>
            <groupId>at.fhj.swd14.pse</groupId>
            <artifactId>backend-interface</artifactId>
        </dependency>
        <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <addClasspath>true</addClasspath>
                        </manifest>
                    </archive>
                    <failOnMissingWebXml>true</failOnMissingWebXml>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

3) Erklären Sie den gegebenen Source Code:

   a. Beschreiben Sie das verwendete **Testprinzip (inkl. Skizze)**.

   b. Erklären Sie das **Page Object Pattern** anhand des gegebenen Beispiels.

```
11 public class PersonUITest extends BaseUITest {
12
13     @BeforeClass
14     public static void setup() {
15         login();
16     }
17
18     @Test
19     public void testProfileLink() {
20         WelcomePage welcomePage = gotoStartPage();
21         welcomePage.changeToOwnProfilePage();
22         boolean isProfilePage = webdriver.getTitle().toLowerCase().contains("my profile");
23         Assert.assertTrue(isProfilePage);
24     }
25
26     @Test
27     public void checkUserList() {
28         UsersPage usersPage = changeToUsersPage();
29         int userCount = usersPage.retrieveUserListCount();
30         Assert.assertTrue(userCount > 0);
31     }
32
33     @Test
34     public void testOtherUserAsContact() {
35         UsersPage usersPage = changeToUsersPage();
36         boolean isFriend = usersPage.addUserAsFriend();
37         Assert.assertTrue(isFriend);
38         boolean isNotFriend = usersPage.removeUserAsFriend();
39         Assert.assertTrue(isNotFriend);
40     }
```

4) Erklären Sie den gegebenen Source Code:

   a. Erklären Sie die **Funktionsweise des Architectural Patterns**, welches hier implementiert wird.

   b. Wozu dient der **Parameter im Konstruktor**?.

   c. Welche **Vorteile** bietet diese abstrakte Basisklasse?

```
10 public abstract class AbstractRepository<T> {
11
12     protected final Class<T> entityClass;
13
14     @PersistenceContext(unitName = "SEP")
15     protected EntityManager entityManager;
16
17     protected AbstractRepository(Class<T> entityClass) {
18         this.entityClass = entityClass;
19     }
20
21     public T find(Long id) {
22         return entityManager.find(entityClass, id);
23     }
24
25     public void update(T t) {
26         entityManager.merge(t);
27     }
28
29     public void save(T t) {
30         entityManager.persist(t);
31     }
32
33     public List<T> findAll() {
34         TypedQuery<T> query = entityManager.createQuery(
35                 "SELECT entity FROM " + entityClass.getTypeName() + " entity", entityClass);
36
37         return query.getResultList();
38     }
39
40     public void refresh(T t) {
41         entityManager.refresh(t);
42     }
43
44     public void remove(T t) {
45         entityManager.remove(t);
46     }
```

5) Erklären Sie die gegebenen Auszüge aus dem **web.xml** File:

   a. Wie funktioniert der **Login Vorgang**?

   b. **Welche Einstellungen** werden in **<security-constraint>** vorgenommen und was bewirken diese?

```
44    <security-constraint>
45        <web-resource-collection>
46            <web-resource-name>secure</web-resource-name>
47            <url-pattern>/*</url-pattern>
48        </web-resource-collection>
49        <auth-constraint>
50            <role-name>user</role-name>
51        </auth-constraint>
52        <user-data-constraint>
53            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
54        </user-data-constraint>
55    </security-constraint>
```

```
85    <login-config>
86        <auth-method>FORM</auth-method>
87        <form-login-config>
88            <form-login-page>/login.xhtml</form-login-page>
89            <form-error-page>/login_failed.xhtml</form-error-page>
90        </form-login-config>
91    </login-config>
```

Practical Software Engineering

SWD14 WS 2016

Name: _____ **Datum: 24.2.2017**

Pro Frage können max. 8 Punkte erreicht werden.
Daraus ergibt sich eine max. Gesamtpunkteanzahl von 40 Punkten.

Punkte: _____

1) Erklären Sie den gegebenen Source Code:

   a) Beschreiben Sie die Funktionsweise des **Architectural Pattern**, welches hier implementiert wird (inkl. **Skizze**).

   b) **Beschreiben Sie die Elemente** des gegebenen Source Codes im Kontext dieses Architectural Patterns.

```
13 @Stateless
14 public class CommentServiceImpl implements CommentService {
15
16     private static final Logger LOGGER = LogManager.getLogger(CommentServiceImpl.class);
17
18     @EJB
19     private CommentRepository commentRepository;
20
21     @Override
22     public long save(CommentDto comment) {
23         LOGGER.trace("Saving comment");
24         try {
25             Comment doComment = CommentConverter.convert(comment);
26             commentRepository.save(doComment);
27             LOGGER.info("Comment {} saved", doComment.getId());
28             return doComment.getId();
29         } catch (Exception e) {
30             LOGGER.warn(e.getMessage(), e);
31             throw new CommentServiceException("Failed to save comment");
32         }
33     }
```

2) Erklären Sie das gegebene **Konfigurationsfile**:

   a. **Wozu** wird dieses Konfigurationsfile verwendet (erklären Sie auch die **verwendeten XML Elemente**)?

   b. **Wo** werden die Einstellungen von **datasources/SEP** konfiguriert?.

```
3  <persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
4      <persistence-unit name="SEP" transaction-type="JTA">
5          <jta-data-source>java:jboss/datasources/SEP</jta-data-source>
6          <properties>
7              <property name="showSql" value="false"/>
8              <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLInnoDBDialect"/>
9          </properties>
10     </persistence-unit>
11 </persistence>
```

3) Erklären Sie den gegebenen Source Code:

   a. Beschreiben Sie das verwendete **Testprinzip (inkl. Skizze)**.

   b. Erklären Sie die verwendeten **Annotations**.

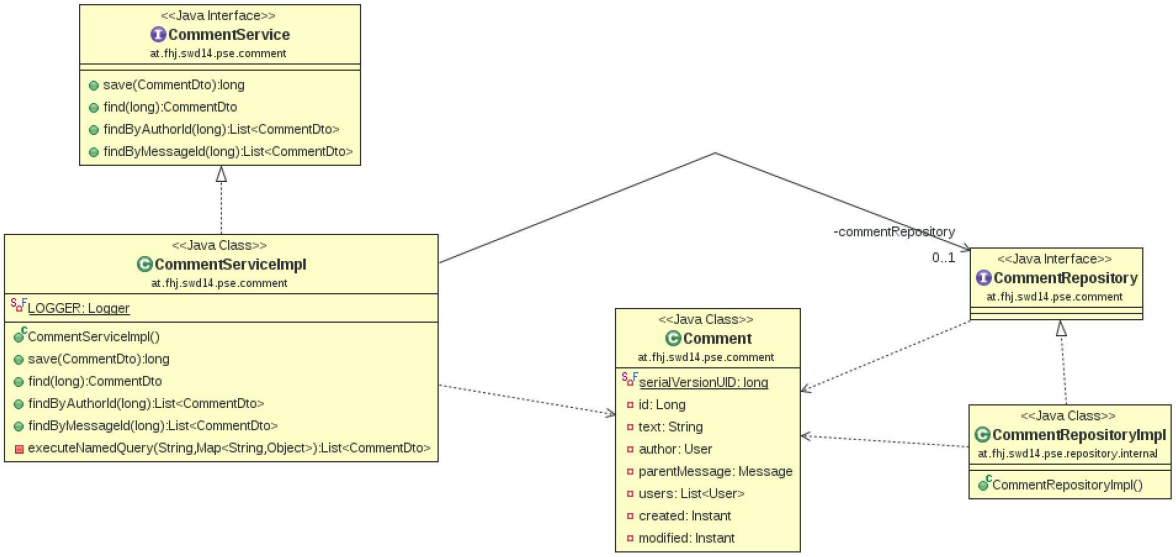
```
22 @RunWith(MockitoJUnitRunner.class)
23 public class CommentServiceImplTest {
24     @InjectMocks
25     private CommentServiceImpl commentServiceImpl;
26
27     @Mock
28     private CommentRepositoryImpl commentRepository;
29
30     private User user;
31     private Message message;
32     private Comment comment;
33     private Community community;
34     private List<Comment> comments;
35     private final Long commentId = 1L;
36     private final Long messageId = 2L;
37     private final Long userId = 3L;
38     private final Long communityId = 4L;
39
41     public void setup() {
60
61
62     @Test(expected = CommentServiceException.class)
63     @SuppressWarnings("unchecked")
64     public void shouldConvertExceptionsWhenFindingASingleComment() {
65         when(commentRepository.find(commentId)).thenThrow(IllegalStateException.class);
66         commentServiceImpl.find(commentId);
67     }
```

4) Erklären Sie das gegebene Class Diagram:

   a) Zeichnen Sie jeweils den **Domain**- und **Data-Source Layer** ein.

   b) Erklären Sie die Richtung der Abhängigkeiten zwischen **CommentServiceImpl**, **Comment**, **CommentRepository** und **CommentRepositoryImpl**.

   c) Welche **Vorteile** ergeben sich aus diesem Design – im Vergleich mit einer klassischen Layered Architecture?



5) Erklären Sie den gegebenen Source Code:

   a) **Wozu** wird eine **Login-Funktionalität** überhaupt benötigt?

   b) Beschreiben Sie **welche Teile** (Browser, Web Server, Applikation, Database) in **welcher Reihenfolge** zusammen spielen müssen, um einen Login durchzuführen.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
<h:head>
    <title>PSE Login</title>
    <h:outputStylesheet name="styles/font/glacialindifference.css" />
    <h:outputStylesheet name="styles/login.css" />
    <link href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-ui.css" rel="stylesheet"></link>
    <link href="http://www.primefaces.org/prime-ui/demo/css/aristo/theme.css" rel="stylesheet"></link>
    <link href="http://www.primefaces.org/prime-ui/demo/css/sh.css" rel="stylesheet"></link>
    <link href="http://www.primefaces.org/css/all.css" rel="stylesheet"></link>
</h:head>
<h:body>
    <h1>Login</h1>
    <h:form id="login" onsubmit="action='j_security_check';" prependId="false">
        <div class="input">
            <p:outputLabel for="j_username" value="Username" />
            <p:inputText id="j_username" maxlength="40" size="20" required="true" />
        </div>
        <div class="input">
            <p:outputLabel for="j_password" value="Password" />
            <p:password id="j_password" maxlength="40" size="20" required="true" />
        </div>
        <div class="submit">
            <p:commandButton id="submit" value="Login" ajax="false"/>
        </div>
    </h:form>
</h:body>
</html>
```

Practical Software Engineering

SWD15 WS 2017

Name: _____ **Datum: 9.2.2017**

Pro Frage können max. 8 Punkte erreicht werden.
Daraus ergibt sich eine max. Gesamtpunkteanzahl von 40 Punkten.

Punkte: _____

1) Erklären Sie den gegebenen Source Code:

   a. Beschreiben Sie das **Architectural Pattern**, welches hier verwendet wird (inkl. **Skizze**).

   b. **Welcher Teil** dieses Architectural Patterns wird vom vorliegenden Source Code implementiert?

   c. **Beschreiben Sie die Elemente** des gegebenen Source Codes im Kontext dieses Architectural Patterns.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    xmlns:b="http://bootsfaces.net/ui" xmlns:p="http://primefaces.org/ui">
<h:head>
    <title>PSE 2017/18</title>
    <meta name="author" content="SWD15"></meta>
    <h:outputStylesheet library="css" name="style.css" />
</h:head>
<h:body>
    <b:navBar brand="PSE 2017/18" fixed="top" inverse="true">
    </b:navBar>
    <b:jumbotron>
        <b:container>
            <h2>Login</h2>
            <b:panel>
                <p:messages id="messages" />
                <h:outputText id="error" value="#{loginBean.errorMsg}"
                    rendered="#{not empty loginBean.errorMsg}" style="..." />
                <b:form id="loginForm">
                    <b:inputText id="username" name="username"
                        value="#{loginBean.username}" label="Username" />
                    <b:inputSecret id="password" name="password"
                        value="#{loginBean.password}" label="Password" />
                    <b:commandButton update=":error" value="Login"
                        action="#{loginBean.doLogin}" ajax="false" />
                </b:form>
            </b:panel>
        </b:container>
    </b:jumbotron>
    <ui:insert name="footer">
        <ui:include src="templates/footer.xhtml" />
    </ui:insert>
</h:body>
</html>
```

2) Erklären Sie den gegebenen Auszug aus dem **pom.xml** File:

   a. **Wozu** wird dieses Plugin verwendet?
   (Erklären Sie die **Funktionsweise** basierend auf den getroffenen Einstellungen)

   b. Nach welchen **Schritten** wird es im **Build Cycle** aufgerufen?

```
43
44          <plugin>
45              <groupId>org.apache.maven.plugins</groupId>
46              <artifactId>maven-failsafe-plugin</artifactId>
47              <version>2.20.1</version>
48              <executions>
49                  <execution>
50                      <id>[redacted]</id>
51                      <phase>integration-test</phase>
52                      <goals>
53                          <goal>integration-test</goal>
54                      </goals>
55
56                      <configuration>
57                          <includes>
58                              <include>**/*DAO*.java</include>
59                          </includes>
60                      </configuration>
61                  </execution>
62
63
64                  <execution>
65                      <id>[redacted]</id>
66                      <phase>post-integration-test</phase>
67                      <goals>
68                          <goal>integration-test</goal>
69                      </goals>
70                  </execution>
71              </executions>
72          </plugin>
```

3) Erklären Sie den gegebenen Source Code:

   a. Beschreiben Sie das verwendete **Testprinzip (inkl. Skizze)**.

   b. Erklären Sie das **Page Object Pattern** anhand des gegebenen Beispiels.

```java
public class FunctionalITCase {
    private LoginPage loginPage;
    private CommunityOverviewPage communityOverviewPage;
    private UserOverviewPage userOverViewPage;
    private ProfilePage profilePage;
    private ActivityStreamPage activityStreamPage;

    private String validUsername = "baar";
    private String validPassword = "pass";

    @Before
    public void setUp() throws Exception {
        loginPage = new LoginPage();
        activityStreamPage = loginPage.login(validUsername, validPassword);
    }

    @Test
    public void testValidLogin() {
        assertEquals( expected: "Activity Stream", activityStreamPage.getHeader());
    }

    @Test
    public void testLogout() {
        activityStreamPage.logout();
        activityStreamPage.refresh();

        assertEquals( expected: "Login", activityStreamPage.getHeader());
    }
}
```

```java
public class LoginPage extends PageObject {

    private String url = "/pse/login.xhtml";

    public LoginPage() { super(); }

    public ActivityStreamPage login(String username, String password) {
        driver.get(baseUrl + url);
        driver.findElement(By.id("input_loginForm:username")).clear();
        driver.findElement(By.id("input_loginForm:username")).sendKeys(username);
        driver.findElement(By.id("input_loginForm:password")).clear();
        driver.findElement(By.id("input_loginForm:password")).sendKeys(password);
        driver.findElement(By.id("loginForm:j_idt11")).click();

        return new ActivityStreamPage(driver);
    }
}
```

4) Erklären Sie den gegebenen Source Code:

   a. Erklären Sie die **Funktionsweise des Architectural Patterns**, welches hier implementiert wird.

   b. Welche **Vorteile** bietet diese Form der **Implementierung**?

   c. Wozu dient die abstrakte Methode **getEntityClass()** ?

```
9   public abstract class DAOImplTemplate<E> implements DAOTemplate<E> {
10      @PersistenceContext
11      protected EntityManager em;
12
13      public EntityManager getEntityManager() {
14          return em;
15      }
16
17      protected abstract Class<E> getEntityClass();
18
19      @Override
20      public E insert(E entity) {
21          em.persist(entity);
22          return entity;
23      }
24
25      @Override
26      public E update(E entity) { return em.merge(entity); }
29
30      @Override
31      public void delete(E entity) { em.remove(entity); }
34
35      @Override
36      public E findById(int id) { return em.find(getEntityClass(), id); }
39
40      @SuppressWarnings("unchecked")
41      @Override
42      public List<E> findAll() {
43          final String hql = "SELECT u FROM " + getEntityClass().getName() + " AS u";
44          return em.createQuery(hql).getResultList();
45      }
46  }
```

5) Sie haben den Source Code Ausschnitt einer **Entity Klasse** gegeben:

   a. Welche **Probleme** erkennen Sie am gegebenen **Exception Handling**?
   (Begründen Sie jedes gefundene Problem)

   b. Wie würden Sie das **Exception Handling verbessern**?
   (Begründen Sie Ihre Verbesserungsvorschläge)

```
65      public void setUser(User user) throws DatabaseException {
66          if (user == null)
67              throw new DatabaseException("User must not be null");
68          this.user = user;
69          try {
70              user.addUserContacts(this);
71          } catch (DatabaseException e) {
72              throw new DatabaseException("setUser() in UserContact: " + user, e);
73          }
74      }
```

Practical Software Engineering

SWD15 WS 2017

Name: _____ **Datum: 27.4.2017**

Pro Frage können max. 8 Punkte erreicht werden.
Daraus ergibt sich eine max. Gesamtpunkteanzahl von 40 Punkten.

Punkte: _____

1) Erklären Sie den gegebenen Source Code:

   a. Beschreiben Sie das **Architectural Pattern**, welches hier implementiert wird (inkl. **Skizze**).

   b. Erklären Sie die gegebene Vorgehensweise beim **Error-Handling** (warum wurde das so implementiert).

```
17  @Stateless
18  public class UserServiceImpl implements UserService {
19      private final Logger LOG = Logger.getLogger(UserServiceImpl.class);
20
21      @Inject
22      private UserDAO userDAO;
23      @Inject
24      private UserContactDAO userContactDAO;
25      @Inject
26      private UserProfileDAO userProfileDAO;
27      @Inject
28      private CommunityDAO communityDAO;
29      @Inject
30      private EnumerationDAO enumDAO;
31
32      @Override
33      public void insert(User user) {
34          LOG.debug("insert " + user);
35          userValidator(user);
36
37          try {
38              userDAO.insert(user);
39          } catch (Exception e) {
40              LOG.error( message: "Can't insert user " + user, e);
41              throw new ServiceException("Can't insert user " + user);
42          }
```

2) Erklären Sie den gegebenen Auszug aus einem Konfigurations-File:

   a. Wozu wird das **persistence.xml** File innerhalb der Applikation benötigt?

   b. Wozu dienen die **verwendeten Elemente** (welche Einstellungen werden hier vorgenommen)?

```xml
<persistence-unit name="pse">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class>org.se.lab.data.Post</class>
    <class>org.se.lab.data.Community</class>
    <class>org.se.lab.data.Enumeration</class>
    <class>org.se.lab.data.User</class>
    <class>org.se.lab.data.UserContact</class>
    <class>org.se.lab.data.UserProfile</class>
    <class>org.se.lab.data.PrivateMessage</class>
    <class>org.se.lab.data.File</class>

    <properties>
        <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/pse" />
        <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver" />
        <property name="hibernate.connection.username" value="student" />
        <property name="hibernate.connection.password" value="student" />
        <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect" />

        <property name="hibernate.show_sql" value="true" />
        <property name="jboss.as.jpa.providerModule" value="org.hibernate:5.0" />
        <property name="eclipselink.ddl-generation" value="create-tables" />
        <property name="toplink.ddl-generation" value="create-tables" />
        <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema" />
        <property name="hibernate.hbm2ddl.auto" value="create" />

        <!--  populate database with sample data at deployment -->
        <property name="javax.persistence.schema-generation.database.action" value="drop-and-create" />
        <property name="javax.persistence.sql-load-script-source" value="META-INF/sql/insert.sql" />
    </properties>

</persistence-unit>
```

3) Erklären Sie den **Unterschied** zwischen den beiden, im pom.xml File verwendeten, Plugins. Gehen Sie auch auf die **Ausführungsreihenfolge** dieser Plugins ein.

   a. maven-surefire-plugin

   b. maven-failsafe-plugin

4) Erklären Sie den gegebenen Source Code:

   a. Beschreiben Sie das **Prinzip von Mock-Objects**.

   b. Erklären Sie die **Setup-Phase** des gegebenen Beispiels.

```
@RunWith(EasyMockRunner.class)
public class CommunityServiceTest {

    public static final int ID = 1;
    public static final String NAME = "testcommunity";
    public static final String DESCRIPTION = "testcommunity description";
    @Rule
    public EasyMockRule mocks = new EasyMockRule( test: this);
    List<Community> communities;
    @TestSubject
    private CommunityService communityService = new CommunityServiceImpl();
    @Mock
    private EnumerationService enumerationService;
    @Mock
    private CommunityDAO communityDAO;
    @Mock
    private FileDao fileDao;
    private Community community1;
    private Community community2;
    private Community community3;

    @Before
    public void setUp() throws Exception {
        Enumeration pending = new Enumeration( id: 1);
        pending.setName("pending");
        Enumeration approved = new Enumeration( id: 2);
        approved.setName("approved");
        Enumeration refused = new Enumeration( id: 3);
        refused.setName("refused");
        expect(enumerationService.getPending()).andStubReturn(pending);
        expect(enumerationService.getApproved()).andStubReturn(approved);
        expect(enumerationService.getRefused()).andStubReturn(refused);
        replay(enumerationService);
        community1 = new Community( name: "name1",  description: "description1", portaladminId: 1);
        community1.setState(enumerationService.getApproved());
        community2 = new Community( name: "name2",  description: "description2", portaladminId: 1);
        community2.setState(enumerationService.getPending());
        community3 = new Community( name: "name3",  description: "description3", portaladminId: 1);
        community3.setState(enumerationService.getRefused());
        communities = new ArrayList<>();
    }
```

5) Erklären Sie den gegebenen Source Code:

   a. Erklären Sie die **Funktionsweise des View-Helper Patterns**, anhand des gegebenen Beispiels.

   b. **Wer instanziiert diese Klasse** und **wie lange lebt** diese Instanz?

```
17  @Named
18  @RequestScoped
19  public class LoginBean implements Serializable {
20      private static final long serialVersionUID = 1L;
21      private final Logger LOG = Logger.getLogger(LoginBean.class);
22      private String username;
23      private String password;
24      private User user;
25      private String errorMsg = "";
26
27      @Inject
28      private UserService service;
29
30      @PostConstruct
31      public void init() {
32
33      }
34
35      public String getUsername() { return username; }
38
39      public void setUsername(String username) { this.username = username; }
42
43      public String getPassword() { return password; }
46
47      public void setPassword(String password) { this.password = password; }
50
51      public void doLogin() {...}
82
83      public String logout() {...}
91
92      public String getErrorMsg() { return errorMsg; }
95
96      public void setErrorMsg(String errorMsg) { this.errorMsg = errorMsg; }
99  }
```

Practical Software Engineering

SWD15 WS 2017

Name: _____ **Datum: 19.5.2017**

Pro Frage können max. 8 Punkte erreicht werden.
Daraus ergibt sich eine max. Gesamtpunkteanzahl von 40 Punkten.

Punkte: _____

1) Erklären Sie den gegebenen Source Code:

   a. Beschreiben Sie das **Architectural Pattern**, welches hier implementiert wird (inkl. **Skizze**).

   b. Erklären Sie die gegebene Vorgehensweise beim **Error-Handling** (warum wurde das so implementiert).

```
17  @Stateless
18  public class UserServiceImpl implements UserService {
19      private final Logger LOG = Logger.getLogger(UserServiceImpl.class);
20
21      @Inject
22      private UserDAO userDAO;
23      @Inject
24      private UserContactDAO userContactDAO;
25      @Inject
26      private UserProfileDAO userProfileDAO;
27      @Inject
28      private CommunityDAO communityDAO;
29      @Inject
30      private EnumerationDAO enumDAO;
31
32      @Override
33      public void insert(User user) {
34          LOG.debug("insert " + user);
35          userValidator(user);
36
37          try {
38              userDAO.insert(user);
39          } catch (Exception e) {
40              LOG.error( message: "Can't insert user " + user, e);
41              throw new ServiceException("Can't insert user " + user);
42          }
```

2) Erklären Sie den gegebenen Auszug aus dem **pom.xml** File:

   a. **Wozu** wird dieses Plugin verwendet?
   (Erklären Sie die **Funktionsweise** basierend auf den getroffenen Einstellungen)

   b. Nach welchen **Schritten** wird es im **Build Cycle** aufgerufen?

```
43
44          <plugin>
45              <groupId>org.apache.maven.plugins</groupId>
46              <artifactId>maven-failsafe-plugin</artifactId>
47              <version>2.20.1</version>
48              <executions>
49                  <execution>
50                      <id>[redacted]</id>
51                      <phase>integration-test</phase>
52                      <goals>
53                          <goal>integration-test</goal>
54                      </goals>
55
56                      <configuration>
57                          <includes>
58                              <include>**/*DAO*.java</include>
59                          </includes>
60                      </configuration>
61                  </execution>
62
63
64                  <execution>
65                      <id>[redacted]</id>
66                      <phase>post-integration-test</phase>
67                      <goals>
68                          <goal>integration-test</goal>
69                      </goals>
70                  </execution>
71              </executions>
72          </plugin>
```

3) Erklären Sie den gegebenen Source Code:

   a. Beschreiben Sie das verwendete **Testprinzip (inkl. Skizze)**.

   b. Erklären Sie das **Page Object Pattern** anhand des gegebenen Beispiels.

```java
public class FunctionalITCase {
    private LoginPage loginPage;
    private CommunityOverviewPage communityOverviewPage;
    private UserOverviewPage userOverViewPage;
    private ProfilePage profilePage;
    private ActivityStreamPage activityStreamPage;

    private String validUsername = "baar";
    private String validPassword = "pass";

    @Before
    public void setUp() throws Exception {
        loginPage = new LoginPage();
        activityStreamPage = loginPage.login(validUsername, validPassword);
    }

    @Test
    public void testValidLogin() {
        assertEquals( expected: "Activity Stream", activityStreamPage.getHeader());
    }

    @Test
    public void testLogout() {
        activityStreamPage.logout();
        activityStreamPage.refresh();

        assertEquals( expected: "Login", activityStreamPage.getHeader());
    }
}
```

```java
public class LoginPage extends PageObject {

    private String url = "/pse/login.xhtml";

    public LoginPage() { super(); }

    public ActivityStreamPage login(String username, String password) {
        driver.get(baseUrl + url);
        driver.findElement(By.id("input_loginForm:username")).clear();
        driver.findElement(By.id("input_loginForm:username")).sendKeys(username);
        driver.findElement(By.id("input_loginForm:password")).clear();
        driver.findElement(By.id("input_loginForm:password")).sendKeys(password);
        driver.findElement(By.id("loginForm:j_idt11")).click();

        return new ActivityStreamPage(driver);
    }
}
```

4) Erklären Sie den gegebenen Source Code:

   a. Erklären Sie die **Funktionsweise des Architectural Patterns**, welches hier implementiert wird.

   b. Welche **Vorteile** bietet diese Form der **Implementierung**?

   c. Wozu dient die abstrakte Methode **getEntityClass()** ?

```
9   public abstract class DAOImplTemplate<E> implements DAOTemplate<E> {
10      @PersistenceContext
11      protected EntityManager em;
12
13      public EntityManager getEntityManager() {
14          return em;
15      }
16
17      protected abstract Class<E> getEntityClass();
18
19      @Override
20      public E insert(E entity) {
21          em.persist(entity);
22          return entity;
23      }
24
25      @Override
26      public E update(E entity) { return em.merge(entity); }
29
30      @Override
31      public void delete(E entity) { em.remove(entity); }
34
35      @Override
36      public E findById(int id) { return em.find(getEntityClass(), id); }
39
40      @SuppressWarnings("unchecked")
41      @Override
42      public List<E> findAll() {
43          final String hql = "SELECT u FROM " + getEntityClass().getName() + " AS u";
44          return em.createQuery(hql).getResultList();
45      }
46  }
```

5) Erklären Sie den gegebenen Source Code:

   a. Erklären Sie die **Funktionsweise des View-Helper Patterns**, anhand des gegebenen Beispiels.

   b. **Wer instanziiert diese Klasse** und **wie lange lebt** diese Instanz?

```
17  @Named
18  @RequestScoped
19  public class LoginBean implements Serializable {
20      private static final long serialVersionUID = 1L;
21      private final Logger LOG = Logger.getLogger(LoginBean.class);
22      private String username;
23      private String password;
24      private User user;
25      private String errorMsg = "";
26
27      @Inject
28      private UserService service;
29
30      @PostConstruct
31      public void init() {
32
33      }
34
35      public String getUsername() { return username; }
38
39      public void setUsername(String username) { this.username = username; }
42
43      public String getPassword() { return password; }
46
47      public void setPassword(String password) { this.password = password; }
50
51      public void doLogin() {...}
82
83      public String logout() {...}
91
92      public String getErrorMsg() { return errorMsg; }
95
96      public void setErrorMsg(String errorMsg) { this.errorMsg = errorMsg; }
99  }
```

a) Welches Architectural Pattern ist es. Skizze + erklären was es macht.
b) Wo wird die Klasse instanziiert und wie lange lebt sie?

```java
@Named
@SessionScoped
public class CreateCommunityController implements Serializable {

    @Inject
    private CommunityService communityService;

    private static final long serialVersionUID = 1L;

    private static final Logger logger = LoggerFactory.getLogger(CreateCommunityController.class);

    private String communityName;
    private boolean isPublic;


    public String getCommunityName() { return communityName; }

    public boolean isPublic() { return isPublic; }

    public void create() { communityService.insertCommunity(getCommunityName(),isPublic); }
}
```

Welches Architectural Pattern

```java
public class RoleMapper {

    public static Role toEntity(RoleDTO roleDTO) {
        Role role = new Role(roleDTO.getName());
        role.setId(roleDTO.getId());
        return role;
    }

    public static RoleDTO toDTO(Role role) {
        RoleDTO roleDTO = new RoleDTO();
        roleDTO.setId(role.getId());
        roleDTO.setName(role.getname());

        return roleDTO;
    }
}
```

Welcher Test ist das, was tut er und welche Vorteile bringt diese Testform. Ich glaub da war auch eine Skizze.

```java
@Test
public void testLoginSuccess() throws Exception
{
    page = new LoginPage(driver,  baseUrl: "http://localhost:8080/chr-krenn-fhj-ws2018-swd16-pse",  timeout: 30);

    // setup
    page.setUsername("admin@swd.com");
    page.setPassword("admin");

    // exercise
    WelcomePage welcome = page.login();
    String userProfileLinkText = welcome.getUserProfileLinkText();

    // verify
    Assert.assertEquals( expected: "admin@swd.com (online)", userProfileLinkText);
}
```

Welches Architectural pattern?

Warum wurde die Exception so verwendet?

```java
@Stateless
public class RoleServiceImpl implements RoleService, Serializable {
    private static final long serialVersionUID = -2342147290467639108L;
    private static final Logger logger = LoggerFactory.getLogger(UserServiceImpl.class);

    @Inject
    private RoleDAO roleDAO;

    @Override
    public List<RoleDTO> getAllRoles() {
        try {
            return roleDAO.findAll().stream().map(RoleMapper::toDTO).collect(Collectors.toList());
        } catch (DaoException e) {
            logger.error("Error loading all roles", e);
            throw new ServiceException("Error loading all roles");
        } catch (Throwable e) {
            logger.error("Unknown error loading all roles", e);
            throw new ServiceException("Unknown error loading all roles");
        }
    }
}
```

Was macht das persistence.xml. Beschreibe die Konfigurationen.

Wo befindet sich das persistence.xml im Projekt

```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/psoDS</jta-data-source>
        <properties>
            <!-- Properties for Hibernate -->
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect" />

            <!--
                SQL stdout logging
            -->
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="use_sql_comments" value="true"/>

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

</persistence>
```