

Leitfaden zum Einzelbeispiel (SEPM/JAVA)

1

Bitte lesen Sie dieses Dokument aufmerksam und bis zum Ende durch, bevor Sie mit der Arbeit beginnen. Der Leitfaden hilft Ihnen bei der korrekten und effizienten Lösung des Einzelbeispiels.

Für alle Projekte gilt, je später mögliche Fehler gefunden werden, umso höher steigen die Kosten für Ihre Behebung. Dies gilt auch für ihr Einzelbeispiel in SEPM. Sollten Sie z.B. einen Fehler erst in Schritt 5 der folgenden Herangehensweise finden, müssen Sie diesen auch in den Schritten 1-4 ausbessern. Achten Sie deshalb darauf, alle Bedingungen korrekt zu erfüllen und lassen Sie Ihre Umsetzung ggf. von einem Kollegen reviewen (Review heißt nicht, die Aufgabe gemeinsam zu machen oder zu kopieren! Denken Sie daran, dass Plagiate negativ bewertet werden!).

Führen Sie Ihre Stundenliste von Anfang an und notieren Sie, wie viel Zeitaufwand Sie bei welcher Teilaufgabe hatten. Trennen Sie hierbei Einarbeitung und tatsächliche Implementierung. Die Stundenliste ist ein Mittel, Ihren Aufwand zukünftig besser abschätzen zu können. Seien Sie deshalb ehrlich zu sich selbst und schreiben Sie wahrheitsgemäße Zeiten auf. Die Beurteilung des Einzelbeispiels hängt nicht mit der Summe der Stunden zusammen, die Sie auf Ihrer Stundenliste notiert haben!

Verwenden Sie das für diese Übung zur Verfügung gestellte Sourcecode Management System (SCM). In der Einzelphase ist dies Subversion (SVN). Sie werden ähnliche Systeme in der Gruppenphase brauchen, da diese ein notwendiges Mittel zur gemeinsamen Entwicklung von Softwareprojekten sind.

Comitten Sie Ihre Arbeit früh und oft („save early, save often“), mindestens jedoch täglich. Verwenden Sie Commit-Messages um sich später zurecht zu finden, falls Sie auf einen früheren Entwicklungsstand zurückkehren müssen. Das SCM ist eine Sicherheit für Sie, bei Hardwareausfall nicht die gesamte Arbeit zu verlieren - verwenden Sie es (Computerausfälle, wenige Tage vor der Deadline, verlieren so einiges an Schrecken)!

Für viele Studierende rückt das Projektmanagement in der Einzelphase in den Hintergrund bzw. es wird kaum über Teilaufgaben nachgedacht und einfach los programmiert. Für die Einzelphase dürfen Sie Ihr Vorgehensmodell selbst wählen, wir empfehlen jedoch, sich an agilen Methoden zu orientieren, um bereits für die Gruppenphase vorzuarbeiten. Die Liste auf der nächsten Seite beschreibt eine mögliche Vorgehensweise, das Endprodukt in Teilprodukte (Alpha, Beta, Final) zu gliedern und die Fertigstellung dieser Teilprodukte bereits in der Einzelphase zu planen.

Die Liste basiert auf Ideen der agilen Entwicklung und ist eine Kombination der Produkt Backlogs¹ und Sprint Backlogs aus SCRUM

¹[http://en.wikipedia.org/wiki/Scrum_\(development\)#Product_backlog](http://en.wikipedia.org/wiki/Scrum_(development)#Product_backlog)

1 Projektplanung & Projektmanagement

1.1 Vorschläge für Releases (Meilensteine)

- **Nach einer Woche. Release 1, Alpha:** „ich kann Code herzeigen“
- **Nach zwei Wochen. Release 2, Beta:** „ich kann das Programm präsentieren“
- **Nach drei Wochen. Release 3, Final:** „ich komme durch die Einzelphase“

1.2 Schritte für die Einzelphase

Dieser Backlog enthält nur technische Arbeitspakete, z.B. jene die in Sourcecode resultieren. Andere Arbeitspakete, wie z.B. die Erstellung der Dokumentation, sollen für die Eingangsphase nur in der Stundenliste aufgezeichnet werden.

Hinweis: Einträge auf dieser Liste, SVN Commit Messages und Einträge auf der Studentenliste, können alle sehr gut kombiniert werden.

Schritt	Beschreibung	Aufwand	Release
3, 4	Datenbank mit allen Tabellen erstellen, auf Papier entworfenes Schema implementieren (CREATE Script)	2	1, Alpha
3, 4	Testdaten erzeugen (INSERT Script)	2	1, Alpha
6.2	Erste JDBC Verbindung zur Datenbank	3	1, Alpha
6.2	Java Main Programm (Hello World)	1	1, Alpha
6.3	Data Transfer Objects erzeugen (beide)	2	1, Alpha
6.4	DAO Interfaces definieren (beide)	2	1, Alpha
6.6	CRUD JUnit Tests für das erste DAO	6	1, Alpha
6.5	Impl. CRUD Methoden für das erste DAO	8	1, Alpha
10	Serviceschicht erstellen (vorhandene JUnit Tests verwenden und daraus Anwendungsfälle implementieren)	4	2, Beta
11.1	GUI Hauptfenster (Menu, Table View, Search)	8	2, Beta
11.1	UI Controller erstellen (an die Serviceschicht und UI anbinden)	4	2, Beta
6.6	CRUD JUnit Tests für das zweite DAO	4	3, Final
6.5	Impl. CRUD Methoden für das zweite DAO	6	3, Final
10	Serviceschicht fertig implementieren, alle Anforderungen abdecken, auch Auswertungen	8	3, Final
10	Optional: Serviceschicht Tests erstellen, vorhandenen Code von den Listener und Event Handler Methoden verwenden	4	3, Final
11.1	GUI fertigstellen (Edit/Insert Fenster sowie Auswertungen)	10	3, Final
11.1	Listener und Event Handler Methoden fertig implementieren	4	3, Final

2 Erstellen eines Domänenmodells

Erstellen Sie mit Hilfe eines geeigneten Programms (z.B.: Dia²) ein Domänenmodell. Wir empfehlen die UML Klassendiagrammnotation. Verwenden Sie dabei dieselbe Notation wie sie in den Lehrveranstaltungen „Datenmodellierung“ und „Datenbanksysteme“ vorgestellt wurde. Achten Sie dabei darauf, dass das Domänenmodell Ihre Sicht/Ihr Verständnis des Datenmodells in der realen Welt darstellt und keine 1:1 Abbildung des Datenbankdesigns ist.

3 Transformation der Entitäten in SQL (1, Alpha)

Überführen Sie die Entitäten in „CREATE TABLE ...“ Statements Ihrer Datenbank. Erstellen Sie hierzu eine Datei, z.B.: create.sql und fügen Sie dort die Statements zum Erstellen und Befüllen der Datenbanktabellen ein.

Ziehen Sie hierzu auch die Dokumentation³ der H2 Datenbank zu Hilfe.

Hinweis: Achten Sie beim Erstellen von database.sql darauf, dass jedes Statement genau in einer Zeile ist. D.h. jedes „CREATE TABLE ...“ bzw. „INSERT INTO ...“ Statement (für die Testdaten) ist genau in einer Zeile und enthält keinen Zeilenumbruch.

Stellen Sie nochmals sicher, dass alle Anforderungen (Datentypen, Primary Keys, Foreign Keys, ...) erfüllt wurden. Sollten Sie Fehler bemerkt haben, korrigieren Sie diese umgehend, auch in der Dokumentation.

4 Erstellen der Datenbank (1, Alpha)

Starten Sie nun die H2 Datenbank. Laden Sie sich dazu die aktuelle Version von H2 herunter und entpacken Sie sie. Öffnen Sie eine Kommandozeile und wechseln nach /bin im H2-Verzeichnis, starten Sie den Datenbankserver mit folgendem Kommando

Listing 1: H2 Server starten, Version: 1.4.181

```
1 java -jar h2*.jar
```

H2 ist nun über das eingebaute Webinterface unter <http://localhost:8082> erreichbar. Sobald Sie sich im Webinterface zu einer Datenbank verbinden, die noch nicht existiert, wird diese erstellt. Jetzt können Sie mit Hilfe ihres sql-Files die benötigten Tabellen erstellen.

Stellen Sie sicher, dass jedes Statement korrekt ausgeführt wurde und die Tabellen erfolgreich angelegt wurden. Überprüfen Sie dies auch mittels geeignetem „SELECT ...“ Statement.

Sollten Sie auf Probleme stoßen, ziehen Sie die Dokumentation zu Hilfe und achten Sie wieder darauf, jedes Statement ohne Zeilenumbrüche einzugeben.

²<https://wiki.gnome.org/Apps/Dia>

³<http://www.h2database.com/html/grammar.html>

5 Erstellen des Klassendiagramms

Erstellen Sie nun mit Hilfe eines geeigneten Tools (z.B.: Dia⁴) das Klassendiagramm Ihrer Applikation.

Achten Sie auf die korrekte Einteilung in Packages. Ihre Applikation wird vermutlich zumindest die folgenden Packages benötigen:

- dao oder persistence
- domain oder entities
- service
- gui

Anmerkung zur GUI-Schicht: Es ist nicht notwendig, jeden Button/Label/Table im Klassendiagramm zu modellieren, sondern „logische“ Elemente (in der Regel Panes, o.ä.) und deren Beziehungen untereinander (vermutlich hierarchisch) und zu den Services (über welche Interfaces die UI-Klassen mit den Service-Klassen kommunizieren) zusammenzufassen.

5.1 Erstellen der Datenzugriffsschicht im Klassendiagramm

Erstellen Sie nun die DAOs (Data-Access-Objects) für Ihre Entitäten.

Die geläufigsten DAO-Operationen sind:

- create
- read/find/search
- update
- delete

Erstellen Sie vorerst die Methoden, die sie sicher brauchen werden.

Informieren Sie sich über den Sinn und Zweck von Data-Access-Objects und die Umsetzung des DAO-Patterns. Stellen Sie sicher, dass Sie verstanden haben, warum das DAO-Pattern verwendet wird und warum es eine Trennung zwischen Interface und Implementierung gibt.

- Interfaces ⁵
- DAO-Pattern ⁶
- Data Transfer Objects ⁷

⁴<https://wiki.gnome.org/Apps/Dia>

⁵<http://docs.oracle.com/javase/tutorial/java/concepts/interface.html>

⁶<http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>

⁷<http://www.oracle.com/technetwork/java/transferobject-139757.html>

6 Umsetzung der Domänenobjekte und der Datenzugriffsschicht in Quellcode 98 99

Wir beginnen nun mit den ersten Schritten der Implementierung und kommen etwas später zur Planung zurück. 100
101

6.1 Einrichten der Entwicklungsumgebung 102

Richten Sie in diesem Schritt nun Ihre Entwicklungsumgebung ein, erstellen Sie ein neues Projekt, fügen Sie SVN-Unterstützung hinzu (z.B. mittels Subclipse für Eclipse oder Tortoise SVN als Standalone Anwendung). 103
104
105

Erstellen Sie die gewünschte Paketstruktur, z.B. 106

```
sepm.xxxx.e0123.dao 107
sepm.xxxx.e0123.domain 108
sepm.xxxx.e0123.service 109
sepm.xxxx.e0123.gui 110
```

wobei xxxx mit aktuellen Semester (z.B. ss15) und e0123 mit Ihrer Matrikelnummer ersetzt wird und checken Sie diese Ordner in SVN ein. 111
112

Erstellen Sie ein /lib Verzeichnis, hier legen Sie alle externen Bibliotheken ab, die für die Entwicklung ihrer Anwendung benötigt werden. Laden Sie log4j2 herunter, kopieren Sie es in das /lib Verzeichnis und fügen es mittels Eclipse zum build-Pfad hinzu, verfahren Sie analog mit allen weiteren Bibliotheken. 113
114
115
116

Achten Sie darauf, dass Sie lediglich die *.jar Dateien hinzufügen und nicht den Quellcode der Bibliotheken. 117
118

6.2 Erstellen der Datenbankverbindung (1, Alpha) 119

Erstellen Sie nun die Verbindung zur Datenbank. Ziehen Sie hierzu auch die Dokumentation Ihrer Datenbank-Engine zu Hilfe. 120
121

Die Verbindung wird über eine Klasse hergestellt, die zumindest die Methoden openConnection(), getConnection() und closeConnection() implementiert und eine Referenz auf die Datenbankverbindung verwaltet. Die Verbindung wird über den entsprechenden Treiber (z.B. jdbc) hergestellt. Diese Klasse, sowie Data Access Objects (DAOs), werden oft als sogenannte Singletons⁸ implementiert. Denken Sie daran, vor der Verwendung, den entsprechenden Treiber zu laden. 122
123
124
125
126
127

Listing 2: H2 Treiber Laden & Verbindungsaufbau

```
1 try {
2   Class.forName("org.h2.Driver");
3 } catch (Exception e) {
4   // Logging & Exceptionhandling 128
5   return;
6 }
7 Connection connection = DriverManager.getConnection("jdbc:h2:tcp://localhost/~/"
   databaseName", "sa", "");
```

WICHTIG: Vergessen Sie nicht die h2.jar in den build-Pfad einzubinden! 129

⁸http://sourcemaking.com/design_patterns/singleton

6.3 Erstellen der Domänenobjekte/Entities (1, Alpha) 130

Erstellen Sie nun im entsprechenden Package Ihre Domänenobjekte. Achten Sie darauf, dass Sie alle Attribute korrekt übernommen haben und entsprechende Getter und Setter erstellt haben. Ihre Entwicklungsumgebung hilft Ihnen bei der Erstellung dieser mittels Kontextmenü. 131
132
133

6.4 Erstellen der DAO-Interfaces (1, Alpha) 134

Erstellen Sie nun im entsprechenden Package die Interfaces zu Ihren DAOs. Best-Practice ist es, das Interface genauso wie das Entity, das zu verwalten ist, zu benennen (z.B.: ProductDAO). 135
136

Versehen Sie jede Methode mit JavaDoc⁹. Stellen Sie sicher, dass jede Methode ausreichend dokumentiert ist und dass das Verhalten bei korrekten sowie bei fehlerhaften Eingabedaten ausreichend spezifiziert ist. 137
138
139

Anmerkung: Mit ausreichend ist gemeint, keine Romane zu Getter/Setter-Methoden zu schreiben und bei DAO & Service- Methoden genaue Spezifikationen anzugeben, d.h. Bedeutung der Parameter und Verhalten im Normal/Fehlerfall. 140
141
142

Stellen Sie sicher, dass sich keine implementierungsspezifischen Methoden im Interface befinden, wie z.B. setDataSource() oder setEntityManager(). Stellen Sie sich hierzu einfach vor, Sie würden ein DAO erstellen, welches die Daten im Arbeitsspeicher hält. Dieses DAO benötigt für gewöhnlich keinen Handler zu einer Datenbank. 143
144
145
146

6.5 Erstellen der DAO-Implementierung (1, Alpha; 2, Beta) 147

Erstellen Sie nun neue Klassen, die die Implementierungen der DAO-Interfaces bilden. 148

Listing 3: Klassendefinition einer JDBC Implementierung eines DAO-Interface

```
1 package sepm.xxxx.e0123.persistence;
2
3 public class JDBCProductDAO implements ProductDAO { /* ... */ }
```

149

Ihre Entwicklungsumgebung wird Ihnen anbieten, die Klasse zu vervollständigen: Nehmen Sie diese Hilfe an, warten Sie dabei noch mit der Implementierung der Funktionalität. 150
151

Sie können bereits jetzt Logging-Statements einfügen, verwenden Sie hierzu die Logging-Bibliothek Log4j¹⁰ 152
153

⁹<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>

¹⁰<http://logging.apache.org/log4j/2.x/>

Listing 4: Klassendefinition einer JDBC Implementierung eines DAO-Interface inklusive Logging

```
1 package sepm.xxxx.e0123.persistence;
2
3 import org.apache.logging.log4j.LogManager;
4 import org.apache.logging.log4j.Logger;
5
6 public class JDBCProductDAO implements ProductDAO {
7
8     private static final Logger LOGGER = LogManager.getLogger();
9
10    /* ... */
11
12    public Product create(Product product) {
13        LOGGER.debug("Entering createMethod with parameters {}", product);
14    }
15
16 }
```

WICHTIG: Vergessen Sie nicht die log4j2-api.jar in den build-Pfad einzubinden und eine entsprechende Konfigurationsdatei anzulegen.

6.6 Erstellen der DAO-Unittests mittels TDD (1, Alpha; 2, Beta)

In diesem Schritt erstellen Sie Unit-Tests für Ihre DAOs. Es ist Best-Practice, eine abstrakte Klasse zu erstellen, die die sogenannten Black-Box-Tests enthält und von dieser Klasse für jede Implementierung des DAOs abzuleiten, um DAO spezifische Parameter zu setzen.

Listing 5: Abstraktes Test-Setup

```
1 package sepm.xxxx.e0123.test;
2
3 public abstract class AbstractProductDAOTest {
4
5     protected ProductDAO productDAO;
6
7     protected setProductDAO(ProductDAO productDAO) {
8         this.productDAO = productDAO;
9     }
10
11     /**
12      * Dieser Test versucht einen ungültigen Wert (NULL-Wert) in die
13      * Datenbank zu speichern. Das DAO sollte eine Exception werfen.
14      */
15     @Test(expected = IllegalArgumentException.class)
16     public createWithNullShouldThrowException() {
17         productDAO.create(null);
18     }
19
20     /**
21      * Dieser Test erstellt ein Objekt des Typs Product mit
22      * korrekten Parametern und speichert es in die Datenbank.
23      */
24     @Test
25     public createWithValidParametersShouldPersist() {
26         // erstelle neues Product-Objekt
27         Product product = new Product( /* ... */ );
28         // hole alle Products
29         List<Product> products = productDAO.findAll();
30         // product darf noch nicht existieren
31         assertFalse(products.contains(product));
32         // speichere product
33         productDAO.create(product);
34         // hole alle products
35         products = productDAO.findAll();
36         // productnt muss nun vorhanden sein
37         assertTrue(products.contains(product));
38     }
39
40     /* ... */
41
42 }
```

161

Listing 6: Ableitung für eine konkrete JDBC Implementierung

```

1 package sepm.xxxx.e0123.test;
2
3 public class JDBCProductDAOTests extends AbstractProductDAOTest {
4
5     private DataSource dataSource;
6
7     @Before
8     public void setUp() throws SQLException {
9         // erstelle DataSource
10        // ggf. Datenbank erstellen (sofern In-Memory und nicht Server-Mode)
11        // Testdaten einspielen
12        // erstelle JDBCProductDAO
13        ProductDAO productDAO = new JDBCProductDAO(dataSource.getConnection());
14        // setze ProductDAO
15        setProductDAO(productDAO);
16        // Starte Transaktion bzw. autoCommit deaktivieren
17        dataSource.getConnection().setAutoCommit(false);
18    }
19
20    @After
21    public void tearDown() throws SQLException {
22        // ein guter Test stellt den Ausgangszustand wieder her, deshalb:
23        // Transaktion zurueckrollen
24        dataSource.getConnection().rollback();
25    }
26
27 }

```

162

Kompilieren Sie die Klassen und führen Sie den JUnit-Test (JDBCProductDAOTest) aus. Alle Tests sollten fehlschlagen, da Sie noch keine Implementierung erstellt haben.

Gehen Sie nun jeden Test durch und erstellen für jeden Test den notwendigen Code in Ihrer Implementierung, sodass der Test nun nicht mehr fehlschlägt.

Diese Vorgehensweise nennt sich **Test-Driven-Development (TDD)**. Sie garantiert, dass jeder Programmcode einen Test besitzt und Sie nur den minimal notwendigen Programmcode erstellen, um der Spezifikation zu genügen. Hier zeigt sich wie wichtig es ist, vollständige und korrekte Dokumentation im Code zu haben. Nur aus einer vollständigen Interface-Dokumentation lassen sich ausreichend viele und gute Unit-Tests erstellen und nur mit Unit-Tests sollte neuer Code in die DAO-Implementierung mit einfließen.

Nach Fertigstellung dieser Aufgabe haben Sie eine funktionierende und getestete Datenzugriffsschicht. Sie können nun beweisen, dass sich Ihre Implementierung im Normalfall korrekt verhält, wenn ungültige Daten übergeben werden. Als kleinen Nebeneffekt haben Sie nun bereits herausgefunden, wie sie sich zu Ihrer Datenbank verbinden können.

7 Erstellen des Anwendungsfalldiagramms/Use-Case Diagramm 177

Erstellen Sie nun, wie in der Lehrveranstaltung „Objektorientierte Modellierung“ oder aus den Vorlesungsfolien zu SEPM beschrieben, das Anwendungsfalldiagramm. Stellen Sie eine korrekte Aktoren-Hierarchie sicher (vermutlich haben Sie nur einen Akteur), achten Sie auf die Systemgrenze und auf extends/include Beziehungen zwischen den Anwendungsfällen. Versetzen Sie sich in die Lage des Anwenders Ihrer Software: Was möchte dieser alles damit tun? Daten einfügen, löschen, ändern, suchen... vielleicht Quartalsberichte erstellen? Berücksichtigen Sie dies in Ihrem Diagramm.

Prüfen Sie Ihr Diagramm gegen diese Angabe: Sind alle Anforderungen (z.B. Auswertungen) erfüllt?

8 Erstellen der Anwendungsfall-Beschreibungen 187

Erstellen Sie nun ausgehend von Ihrem Anwendungsfalldiagramm die Beschreibungen zu Ihren Anwendungsfällen. Ziehen Sie hierzu die Vorlesungsfolien zu Rate. Achten Sie auf korrekte Vor- und Nachbedingungen und auf Alternativszenarien.

In einem Projekt mit mehreren Mitgliedern ist es sehr wichtig, dass die Anwendungsfälle und die Beschreibung möglichst vollständig und fehlerfrei sowie zu einem möglichst frühen Zeitpunkt im Projekt vorliegen, da aus Ihnen gewöhnlich Domänenmodell, ER-Diagramm, Klassendiagramm usw. abgeleitet werden. Auch werden Anwendungsfälle verwendet, um Aufgaben an Programmierer zu verteilen. Ist ein Anwendungsfall ungenau spezifiziert, muss der Programmierer ständig das restliche Team befragen, oder selbst Annahmen treffen, die womöglich nicht im Sinn des ganzen Teams sind. Änderungen an den Anwendungsfällen, nachdem die Arbeit an den anderen Teilen bereits begonnen wurde, sind deshalb besonders kostspielig.

9 Klassendiagramm um Anwendungsfälle erweitern 199

Nachdem die Anwendungsfallbeschreibung fertig gestellt ist, kann das Klassendiagramm um die sogenannte „Service-Schicht“ erweitert werden. Die Service-Schicht enthält für jeden Anwendungsfall für gewöhnlich zumindest eine Methode, die die Logik dieses Anwendungsfalls beinhaltet und die Arbeit verrichtet.

Auch hier empfiehlt es sich wieder zwischen Interface und Implementierung zu trennen, um nicht implementierungsspezifische Details nach oben zu offenbaren (z.B. sollte die GUI nie direkten Zugriff auf die Datenbank mittels DataSource oder Connection haben).

Das Erstellen einer eigenen Schicht erlaubt ausserdem die leichtere Wiederverwendung der gesamten Anwendungslogik, z.B. als Webservice oder für eine Konsolenapplikation.

Erzeugen Sie also in Ihrem Klassendiagramm im Package Service ein Interface als auch eine Implementierung, wobei das Interface für jeden Anwendungsfall eine Methode enthält.

Sie können so Ihre komplette GUI-Anwendung durchplanen, ohne eine einzige Zeile GUI-Code ausprogrammiert zu haben.

Beim Durchdenken der Methoden entdecken Sie vermutlich, dass Sie weitere Methoden in Ihren DAOs benötigen. Überlegen Sie sich gut, was Sie alles brauchen, fügen Sie diese im Klassendiagramm bei den DAOs hinzu. Erweitern Sie sodann die DAO-Interfaces, DAO-Tests als auch die DAO Implementierungen um die entsprechenden Methoden. Achten Sie bereits jetzt darauf, Änderungen konstant überall durchzuziehen, Sie ersparen sich damit das spätere Durchgehen des gesamten Projektes um Dokumentation und Code zu synchronisieren.

10 Umsetzung der Serviceschicht in Code(2, Beta; 3, Final)

219

Erstellen Sie nun aus dem Interface in Ihrem Klassendiagramm ein Java Interface

220

Listing 7: Interface für ein Service

```

1 package sepm.xxxx.e0123.service;
2
3 /**
4  * Dieses Interface stellt alle Methoden gemaess
5  * der Anwendungsfallbeschreibung
6  */
7 public interface Service {
8
9  /**
10   * Diese Methode erstellt einen neuen Product, bei ungueltigen
11   * Daten wird eine IllegalArgumentException geworfen.
12   */
13   public Product createProduct(Product product) throws IllegalArgumentException;
14
15   /* ... */
16
17   /**
18   * Diese Methode fuehrt eine Auswertung gemaess
19   * den gegebenen Parametern durch.
20   */
21   public Report createReport(ReportParameters params);
22
23 }
```

221

Die Implementierung dazu enthält nun auch die notwendigen Ressourcen, um die Aufgabe zu erfüllen

222

223

Listing 8: Implementierung des zuvor erstellten Interface

```

1 package sepm.xxxx.e0123.service;
2
3 public class SimpleService implements Service {
4
5   private ProductDAO productDAO;
6   private CustomerDAO customerDAO;
7
8   // getter + setter fuer DAOs
9
10  public Product createProduct(Product product) throws IllegalArgumentException {
11    return productDAO.create(product); // hier wird einfach nur delegiert
12  }
13
14  /* ... */
15
16  public Report createReport(ReportParameters params) {
17    Product product = productDAO.find(params.product);
18    // komplexere Operation die ggf. ueber mehrere DAO's + Services
19    // hinweg operiert
20    return report;
21  }
22
23 }
```

224

Erstellen Sie wie bei den DAOs eine abstrakte Test-Klasse und eine Ableitung davon für Ihr SimpleService. Nachdem die meisten Methoden Ihrer Service-Schicht vermutlich nur an die DAOs delegieren, können Sie Ihrer Service-Schicht im Test auch konkrete DAOs zuweisen (setProduct-DAO) um zu prüfen, dass diese die Befehle korrekt weiterleiten.

Erstellen Sie zumindest für die Methoden, die nicht einfach nur an die DAOs weiterdelegieren, Tests (die DAOs selbst wurden ja schon getestet), die diese Methoden auf korrektes Verhalten bei gültigen und ungültigen Daten prüfen. Für Auswertungen kommen Ihnen spätestens jetzt die ganz am Anfang erstellten Testdaten zugute.

Nach Fertigstellung dieser Aufgabe haben sie eine lauffähige, gut getestete Software, die bereits alle Anwendungsfälle abdeckt und Daten korrekt persistieren und abrufen kann. Durch die Verwendung eines Interfaces können Sie den bisherigen Code für eine Vielzahl von Benutzeroberflächen verwenden. In dieser Übung werden Sie eine GUI mittels JavaFX erstellen. Sie könnten Ihr Service aber genauso gut als Webservice exportieren oder eine Konsolenapplikation daraus machen, ohne bestehende Funktionalität neu programmieren zu müssen.

Achtung: Vergessen Sie auch hier nicht, entsprechende Logging-Statements in Ihren Code einzuführen.

11 Planen der grafischen Benutzeroberfläche

11.1 Erstellen der Grafischen Benutzeroberfläche (2, Beta; 3, Final)

Wenn Sie bei diesem Punkt angelangt sind, ist Ihr Programm bereits bis auf die Implementierung der grafischen Benutzeroberfläche komplett fertig.

Sehen Sie sich nochmals die Folien vom Java Tutorial im TUWEL an und machen Sie sich mit der Dokumentation von JavaFX¹¹ vertraut.

Achten Sie darauf, dass alle Elemente Ihrer GUI, die Zugriff auf das Service benötigen, auch eine Membervariable für das entsprechende Service haben und dieses von der erstellenden Komponente gesetzt bekommt.

Sorgen Sie dafür, dass nur gültige Daten in Formularfelder eingegeben werden können und das Formular nur abgeschickt werden kann, wenn die Daten gültig sind. Prüfen Sie Ihre Eingabedaten bereits in der GUI.

Erinnern Sie sich an Datenbankmodellierung/-systeme: Primärschlüssel sind ... „Eindeutig und Unveränderbar“. Berücksichtigen Sie dies in Ihrer GUI.

Wir empfehlen an geeigneten Stellen das Hinzufügen von Kontextmenüs. Geben Sie dem User die Möglichkeit Zeilen zu markieren und mittels Rechtsklick zu bearbeiten und zu löschen.

Für das Erstellen der GUI dürfen Sie den grafischen Editor von Oracle, den JavaFX SceneBuilder¹², verwenden.

¹¹<http://docs.oracle.com/javase/8/javase-clienttechnologies.htm>

¹²<http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html>

12 Zusammenstellung der Abgabe

259

Sofern Sie dieser Anleitung gefolgt sind, ist Ihr Projekt nun fertig. Drucken Sie Ihre Dokumentation aus und heften Sie diese. Vergessen Sie Ihre GUI-Skizzen dabei nicht! Checken Sie die letzten Änderungen in das SCM System ein.

260

261

262

Überprüfen Sie auf einem anderen Computer, ob das Auschecken Ihres Sourcecodes das komplette Projekt (inklusive Dokumentation) wieder herstellt, der Sourcecode kompiliert, die Tests nicht fehlschlagen und Ihr Programm funktioniert. Diese Schritte wird der Tutor beim Abgabegespräch auch durchführen, stellen Sie also sicher, dass Ihr Projekt vollständig und funktionstüchtig ist. Vergessen Sie nicht, sich rechtzeitig für ein Abgabegespräch anzumelden.

263

264

265

266

267

Viel Spaß und Erfolg bei der SE&PM Laborübung!

268

13 Weiterführende Literatur	269
13.1 Literatur und Links aus dem Leitfaden	270
- Springer; 2010;	271
Best Practice Software-Engineering; Schatten, Biff, Demolsky, Gostischa-Franta,	272
Östreicher, Winkler	273
- Product Backlog (http://en.wikipedia.org/wiki/Scrum_(software_development)#Product_backlog)	274 275
- Dia (https://wiki.gnome.org/Apps/Dia)	276
- H2 (http://www.h2database.com/)	277
- DAO (http://www.oracle.com/technetwork/java/dataaccessobject-138824.html)	278
- DTO (http://www.oracle.com/technetwork/java/transferobject-139757.html)	279
- Interface (http://docs.oracle.com/javase/tutorial/java/concepts/interface.html)	280
- Singleton (http://sourcemaking.com/design_patterns/singleton)	281
- JavaDoc (http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html)	282 283
- Log4J2 (http://logging.apache.org/log4j/2.x/)	284
- JavaFX (http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html)	285 286

13.2 Literatur und Links für besonders Interessierte (keine Pflicht!)	287
- Design Patterns	288
- Sourcemaking (http://sourcemaking.com/)	289
- Inversion of Control	290
- Dependency Injection (http://en.wikipedia.org/wiki/Dependency_injection)	291
- Spring (http://spring.io/)	292
- Google Guice (https://code.google.com/p/google-guice/)	293
- JSR 330 (http://jcp.org/en/jsr/detail?id=330)	294
- OO-Design-Prinzipien (http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod)	295
- Testing	296
- JUnit (http://junit.org/)	297
- Mocking (http://en.wikipedia.org/wiki/Mock_object)	298
- Mockito (https://code.google.com/p/mockito/)	299
- Google Testing Blog (http://googletesting.blogspot.co.at/)	300
- Clean Code	301
- FindBugs (http://findbugs.sourceforge.net/)	302
- Checkstyle (http://checkstyle.sourceforge.net/)	303
- 2009; Clean Code: A Handbook of Agile Software Craftsmanship	304
- Projekt-LifeCycle und Dependency-Management; Robert C. Martin	305
- Apache Maven (http://maven.apache.org/)	306
- Repository Suche (http://mvnrepository.com/)	307
- Vorträge und Vorlesungen an der TU	308
- 183.239/188.410; Software Engineering und Projektmanagement; VO	309
- 180.764; Software-Qualitätssicherung; VU	310
- Java Student User Group (alle 3 Wochen!) (http://jsug.at/)	311
- Entwicklungsumgebungen	312
- Eclipse (http://www.eclipse.org/)	313
- NetBeans (http://netbeans.org/)	314
- IntelliJ IDEA (http://www.jetbrains.com/idea/)	315