Technology
Arts Sciences
TH Köln

SWP Vorgaben

Softwarepraktikum

TH Köln - Informatik Labor

Version 9.1, 2021-05-04

Inhaltsverzeichnis

1. Zweck des Dokuments

Dieses Dokument erläutert die verbindlichen Vorgaben im Software-Praktikum. Die korrekte Umsetzung dieser Vorgaben geht in die Note zum Software-Praktikum ein. Die Vorgaben sollen eine qualitäts-orientierte Umsetzung der System-Spezifikation und des Grobentwurfs sicherstellen.

2. Vorgaben zu den Werkzeugen

Es sollen die folgenden Versionen der Werkzeuge verwendet werden:

- IntelliJ IDEA Community Edition in Version 2021.x (Download unter https://www.jetbrains.com)
- JDK in Version 17, wir verwenden und empfehlen das Adoptium OpenJDK (https://adoptium.net/)
- JavaFX für die Erstellung der Benutzeroberflächen.
- JUnit in Version 4

Zur Erstellung von UML-Diagrammen empfehlen wir:

- PlantUML: (https://www.plantuml) erstellt UML Diagramem aus Textdateien
- ein gutes freies Programm (online und Desktop-Version): draw.io (https://www.draw.io)
- sehr einfaches Werkzeug: violetUML in Version 2.0.1 (siehe https://violet.sourceforge.net)
- MS Visio
- ein professionelles Werkzeug: Visual Paradigm (Software ist beim Labor für Informatik erhältlich)

3. Verwendung von Gitlab Issue Tracker

Kommentare zu den Meilensteinen MS2, MS3, und MS4 und weitere Kommentare werden von uns als Issues im Gitlab Issue Tracker kommuniziert. Die Teams müssen diese Fehler bis zum nächsten Meilenstein korrigieren, d.h.

- alle Fehler, die vor dem MS3-Termin dokumentiert wurden, müssen bis MS3 behoben werden (also insbesondere die Fehler der Abgaben zu MS2).
- alle Fehler, die nach dem dem MS3-Termin und vor dem MS4-Termin dokumentiert wurden, müssen bis MS4 behoben werden.
- alle nach dem dem MS4-Termin dokumentierten Fehler müssen zum MS5-Termin behoben sein.
- Nach jeder Fehlerkorrektur muss
  - ein neuer Push in den Master-Branch des Git Remote Repository erfolgen,
  - die modular-Jar neu erzeugt werden und im Maven-Repo des SWP zur Verfügung gestellt werden!

Bei der Korrektur soll die Phase der Fehlerbehebung durch den Status des Issues in Gitlab dokumentiert werden. Der Lebenszyklus eines Issues in Gitlab soll im Software-Praktikum folgendermaßen verlaufen:

- Ein neuer Issue erhält von uns den Status `open`.
- Handelt es sich bei diesem Issue um
  - einen Fehler, so wird er mit dem Label `bug` gekennzeichnet,
  - einen Vorschlag oder Hinweis (der keinen Fehler darstellt), so wird er mit dem Label `suggestion` gekennzeichnet. Diese Vorschläge müssen nicht wie Fehler zwingend umgesetzt bzw. korrigiert werden, wir empfehlen es aber.
- Wenn ein Fehler korrigiert wurde, so soll derjenige, der den Fehler korrigiert hat, das Issue in den Status `closed` setzen.
- Falls das Team feststellt, dass ein eingetragener Fehler kein wirklicher Fehler ist oder es sich um eine Duplette handelt, so soll dieser Fehler durch das Team mit dem Label `nobug` gekennzeichnet werden. Zusätzlich soll eine Erklärung hierzu bei dem Issue eingetragen werden.
- Wir kontrollieren regelmäßig den Zustand der Fehler:
  - Falls wir mit der Bearbeitung eines Fehlers im Status `closed` zufrieden sind, dann setzen wir den Fehler auf den Status `approved`. Damit ist die Bearbeitung des Fehlers für das Team erfolgreich abgeschlossen.
  - Falls wir nicht zufrieden sind mit der Lösung des Fehlers, dann kennzeichnen wir den Fehler mit dem Label `reopened` und das Team muss den Fehler erneut bearbeiten und nach Bearbeitung in den Status `closed` setzen.

4. Verwendung von Git

Gitlab wird zur Versionsverwaltung im Software-Praktikum verwendet.

In Git erfolgt auch die Abgabe zu den Meilensteinen MS2, MS3, MS4, MS5. Zu jeder Abgabe (außer MS2) sollen Sie einen entsprechenden Tag setzen: MS3, MS4, MS5. Diese Tags sollen im Git Remote Repository, Branch master erstellt werden, nicht in einem anderen Branch.

Ihre eigenen Branches dürfen nicht die Bezeichnungen MS2, MS3, MS4 oder MS5 besitzen!

Bei Fragen zu Git oder Problemen mit Git, können sich die Teilnehmer an die Tutoren oder die wissenschaftlichen Mitarbeiter wenden.

5. Allgemeine Hinweise

5.1. Schichten-Architektur und Kopplung

Das zu erstellende System im Software-Praktikum besitzt eine geschlossene Schichten-Architektur mit 3 Schichten: Datenhaltung, Fachlogik, Benutzeroberfläche.

Zwischen den Schichten Datenhaltung und Fachlogik muss eine lose Kopplung realisiert werden, d.h.

- Pakete mit den abstrakten Schnittstellen-Klassen der Datenhaltung sollen exportiert und von den Modulen der Fachlogik importiert werden.
- Pakete mit den konkreten Implementierungsklassen der Schnittstellen der Datenhaltung dürfen nicht exportiert werden.
- Die Module der Fachlogik müssen die Implementierungen der Schnittstellen der Datenhaltung mit Hilfe der Klasse `ServiceLoader` ermitteln.
- Es dürfen die Pakete eines Moduls nicht mit `opens` und es darf das Modul nicht mit `open` für Modul-externe Zugriffe geöffnet werden.

Zwischen den Schichten Fachlogik und Benutzeroberfläche soll eine enge Kopplung realisiert werden, d.h.

- Pakete mit den abstrakten Schnittstellen-Klassen der Fachlogik sollen exportiert und von den Modulen der Benutzeroberfläche importiert werden.
- Pakete mit den konkreten Implementierungsklassen der Schnittstellen (also die Steuerungsklassen) der Fachlogik sollen exportiert und von den Modulen der Benutzeroberfläche importiert werden.
- Es dürfen die Pakete eines Moduls nicht mit `opens` und es darf das Modul nicht mit `open` für Modul-externe Zugriffe geöffnet werden.

5.2. IntelliJ-Projekte und Maven-Repository des SWP

Jede Komponente eines Systems (Hinweis: Alle Komponenten eines Systems sind im Grobentwurf definiert.) ist als ein separates IntelliJ-Projekt realisiert. Der Name dieses IntelliJ-Projekts in Gitlab entspricht dem Namen der Komponente (alle Buchstaben sind klein geschrieben).

5.3. Paketnamen und Modulnamen

Bei der Realisierung von Modulen organisieren Sie den Code intern in Paketen, genau wie bei jedem anderen Java-Projekt. Hierbei sollen Sie folgende allgemeine Namenskonvention beachten: Ein Modulname beginnt mit dem umgekehrten Internet-Domänennamen des Unternehmens oder der Organisation. In unserem Fall ist dieses `de.thkoeln`. Dieser Name wird erweitert um Angaben zur zugehörigen Komponente aus dem Grobentwurf (Bsp.: `swp.bks.admindaten`). Ein derart gebildeter Modulname definiert auch die Namen der in ihm vorhandenen Pakete: Alle Paketnamen besitzen den Namen des Moduls als Präfix, d.h. in einem Modul mit dem Namen

`de.thkoeln.swp.bks.admindaten` beginnen alle Pakete mit `de.thkoeln.swp.bks.admindaten` und erweitern diesen gegebenenfalls um weitere Ebenen. Weil alle Modulnamen in einem System eindeutig sein müssen, sind somit auch alle Paketnamen eindeutig. Weiterhin kann für jedes Paket sehr leicht ermittelt werden, in welchem Modul es sich befindet.

6. Hinweise zu MS2

Bis zum MS2 ist zu machen:

- Implementierung der Schnittstellen-Klassen in der Schicht Datenhaltung
  - Das Dokument Arbeitspakete MS2 gibt an, welcher Teilnehmer welche Methoden welcher Schnittstellen-Klasse implementieren soll.
- Implementierung der Testfälle aus der Testspezifikation für die Methoden der implementierten Schnittstellen-Klassen.

Verbindlicher Termin, konkrete Liefergegenstände und genaue Abgabeorte sind in den Folien zur Einführungsveranstaltung angegeben.

6.1. Datenbank-Zugriffe

In den Unterkapiteln des Abschnitt 9.1 finden Sie Informationen für den Zugriff auf die Datenbank unter Verwendung der Gateway-Klasse `IDatabaseImpl`.

6.2. JUnit-Testfälle

- Die JUnit-Testfälle werden entsprechend der Testspezifikation erstellt. Testspezifikationen existieren für alle Schnittstellen der Komponenten der Schicht "Datenhaltung".
- Ein Testfall darf nicht abhängig von der aktuellen Realisierung der Konstruktoren sein. Falls beispielsweise die Testspezifikation fordert "UND die ID des Testkontos gleich NULL ist", dann muss die ID des für den Testfall erzeugten Objekts explizit auf NULL gesetzt werden. Man darf sich nicht darauf verlassen, dass der Konstruktor dieses übernimmt
  - auch wenn dieses in der aktuellen Implementierung erfolgt.
- Der Text des Testfalls in der Testspezifikation wird zum Java-Kommentar des JUnit-Testfalls, damit man den Zusammenhang leicht erkennen kann. Beispiel:

```
/* @Test: insertKonto_00()
WENN die Methode insertKonto mit  einem Testkonto aufgerufen wird,
UND die ID des Testkontos gleich NULL ist,
DANN sollte sie TRUE zurueckliefern,
UND das Testkonto sollte in der DB existieren.
*/
@Test
public void insertKonto_00()
...
```

- Der Zustand der Datenbank soll durch die Ausführung der Testfälle nicht persistent verändert werden, d.h. es darf kein `commit` erfolgen, sondern es muss in der `@After`-Methode ein `rollback` vorgenommen werden.
- Die Testfälle dürfen keine Annahmen über den Zustand der Datenbank machen, da alle

Entwickler auf der gleichen Datenbank entwickeln und sich somit der Zustand der Datenbank ständig verändern wird. Jeder einzelne Testfall muss vielmehr den gewünschten Zustand der Datenbank selbst herstellen.

- Im Software-Praktikum sollen Sie nicht nur funktionierenden Code erstellen, sondern der erstellte Code soll auch möglichst professionell erstellt werden. Wir achten also bei der Überprüfung des Codes nicht nur darauf, dass der Code funktional korrekt ist, sondern auch darauf, dass er sauber, verständlich, durchdacht, effizient erstellt wurde.

  Beispiel: Soll in JUnit beispielsweise geprüft werden, dass eine Variable `result` den Wert `false` besitzt, dann ist folgender Code zwar funktional korrekt, jedoch weder verständlich noch durchdacht: `AssertNotEquals(true,result)`

  Verständlich und durchdacht ist dagegen die folgende Überprüfung der Variable `result`: `AssertFalse(result)`

7. Vorgaben zur Arbeit nach MS2

Die Datenhaltung sollte eigentlich zu MS2 korrekt implementiert sein. Leider enthält die Datenhaltung aber nach MS2 immer noch Fehler, die zumeist bei der Implementierung der Anwendungsfälle entdeckt werden.

Wird bei der Implementierung der Anwendungsfälle ein Fehler in der Datenhaltung entdeckt, so muss für diesen Fehler im Gitlab Issue Tracker ein Bug erzeugt werden.

Der verantwortliche Implementierer der Datenhaltungs-Methode, die dieser Fehler betrifft, muss diesen Fehler innerhalb einer Woche beheben - oder als unbegründet kennzeichnen. Findet durch den verantwortlichen Implementierer keine fristgerechte Bearbeitung des Fehlers statt, so muss sich der Erzeuger des Bugs im Gitlab Issue Tracker bei den Organisatoren des SWP melden (Assis oder Prof).

Es sollten also alle Teams rechtzeitig vor MS4 mit der Implementierung der Anwendungsfälle beginnen, damit Sie Fehler in den verwendeten Datenhaltungs-Methoden rechtzeitig entdecken. Nur dadurch können Sie sicherstellen, dass Ihre Abgabe zu MS4 auch wirklich lauffähig ist. Ist die Abgabe zu MS4 aufgrund zu spät entdeckter Fehler in den verwendeten Datenhaltungs-Methoden nicht lauffähig, so liegt die Schuld hierfür bei den Implementierern der Anwendungsfälle.

8. Hinweise zu MS3

Bis zum MS3 ist zu machen:

- Klassendiagramm mit den vom Team geplanten Klassen der Fachlogik-Schicht und deren Abhängigkeiten
- Anwendungsfall-Tabelle
- konzeptueller Entwurf der GUI für die zu realisierenden Anwendungsfälle
- Korrektur aller Bugs und Setzen des korrekten Status, inkl.
  - ein neuer Push in den Master-Branch des Git Remote Repository.

Das Dokument Arbeitspakete MS4 gibt an, welcher Teilnehmer welche Anwendungsfälle implementieren soll.

Verbindlicher Termin, konkrete Liefergegenstände und genaue Abgabeorte sind in den Folien zur Einführungsveranstaltung und in den Folien vom MS2 angegeben.

8.1. Klassendiagramm

Das zu MS3 geforderte Klassendiagramm ist für die Komponenten der Schicht "Fachlogik" zu erstellen. Für Komponenten der Schicht "Benutzeroberfläche" sind keine Klassendiagramme zu erstellen. Das Klassendiagramm soll neben den Inhalten der Komponenten der Schicht "Fachlogik" auch die jeweiligen "use"-Beziehungen zu den Schnittstellen-Klassen der Komponenten der Schicht "Datenhaltung" enthalten. Diese Schnittstellen-Klassen müssen jedoch nur mit ihrem Namen dargestellt werden, die Angabe der Methoden ist nicht erforderlich.

Hinweis: Das Klassendiagramm ist während des gesamten Software-Praktikums aktuell zu halten, d.h. es muss immer dem aktuellen Code entsprechen. Eine angepasste Version des Klassendiagramms muss zu MS4 abgegeben werden.

8.1.1. Gestaltung des Klassendiagramms

Folgende Hinweise sind bei der Gestaltung eines Klassendiagramms zu befolgen:

- Das Klassendiagramm folgt dem UML-Standard und ist im pdf-Format abzugeben.
- Die Komponente ist in Pakete strukturiert und alle Klassen befinden sich in einem Paket. Die zu verwendenden Paketbezeichnungen sind im Dokument zum Grobentwurf enthalten.
- Klassennamen beginnen mit einem Großbuchstaben
- Attributnamen beginnen mit einem Kleinbuchstaben
- Methodennamen beginnen mit einem Kleinbuchstaben
- Parameternamen sind aussagekräftig (nicht a, b, c usw.)
- Assoziationen besitzen einen Namen und der Name beginnt mit einem Kleinbuchstaben
- Grenzklassen besitzen den Stereotyp «Grenzklasse»

- Steuerungsklassen besitzen den Stereotyp «Steuerungsklasse»
- Schnittstellenklassen besitzen den Stereotyp «Interface»
- Abhängigkeitsbeziehungen (use-Beziehungen in UML) existieren zwischen zwei Klassen, wenn die eine (abhängige) Klasse eine Methode der anderen (unabhängigen) Klasse verwendet. Use-Beziehungen müssen nur von Steuerungsklassen zu anderen Steuerungsklassen und zu Schnittstellen-Klassen eingetragen werden.
- Realisierungsbeziehungen existieren zwischen Implementierungs- und Schnittstellenklassen.
- Attribute haben einen sinnvollen Datentyp
- Attribute sind privat, falls es keine guten Gründe für eine andere Sichtbarkeit gibt
- Methoden haben einen Ergebnistyp und alle Parameter haben einen Datentyp
- Entitätsklassen müssen im Klassendiagramm nicht aufgeführt werden.

8.1.2. Verwendung einer geschlossenen 3-Schichten Architektur

Alle Systeme im Software-Praktikum sollen in einer geschlossenen 3-Schichten-Architektur umgesetzt werden. Es sollen die drei Schichten Benutzeroberfläche, Fachlogik und Datenhaltung existieren.

Wir wollen weiterhin eine größtmögliche Unabhängigkeit zwischen der GUI und der Datenhaltung herstellen, damit eine Änderung an der Datenbank (z.B. neue Attribute, Aufteilung/Zusammenlegung von Tabellen) nicht zu Änderungen an der GUI führt. Diese Unabhängigkeit soll durch eine strenge Entkopplung der GUI-Klassen von der Datenbank mit Hilfe von Grenzklassen erfolgen:

- Die GUI kennt keine Entitätsklassen. Sie darf also von den Steuerungsklassen niemals Objekte der Entitätsklassen erhalten.
- Die gesamte Kommunikation zwischen GUI und Steuerungsklassen findet mit Hilfe von Grenzklassen statt.

Beispiel 1

Daten eines Kunden holen: Die GUI übergibt den Suchparameter (z.B. Name oder ID) an eine Operation der Steuerungsklasse. Diese holt die gewünschten Kundendaten aus der Datenbank, kopiert diese in ein Objekt der Grenzklasse zum Kunden und übergibt dieses an die GUI als Antwort zurück. Somit muss die GUI die Entitätsklassen nicht kennen und kann ausschließlich mit Grenzklassen arbeiten. Eine Änderung der Entitätsklassen erfordert somit auch keine Änderung der GUI-Klassen, sondern nur der Steuerungsklassen. Hierdurch erlangt man eine weitgehende Entkopplung der GUI von der Datenhaltung.

Beispiel 2

Neuen Kunden anlegen: In einem Formular in der GUI gibt der Benutzer die erforderlichen Daten zum neuen Kunden ein. Die GUI speichert diese eingegebenen Daten in einem Objekt der Grenzklasse zum Kunden und übergibt dieses Objekt als Parameter an die entsprechende Operation der Steuerungsklasse. Diese Operation erstellt ein neues Objekt der Entitätsklasse zum Kunden, trägt dort die Daten aus der Grenzklasse ein und persistiert das Objekt der Entitätsklasse. Auch in diesem Fall muss die GUI nicht die konkrete Struktur der

Entitätsklasse kennen und arbeitet lediglich mit der auf ihre Bedürfnisse zugeschnittenen Grenzklasse.

8.2. Anwendungsfall-Tabellen

Eine Vorlage der Anwendungsfall-Tabelle existiert in Ilias.

Die Anwendungsfall-Tabelle enthält alle Anwenungsfälle eines Teams. Für jeden Anwendungsfall ist anzugeben, durch welche Methode welcher Steuerungsklasse er implementiert wird.

8.3. Konzeptueller Entwurf der GUI

Der konzeptuelle Entwurf der GUI hilft dem Kunden zu verstehen, ob die geplante Benutzeroberfläche geeignet ist, die gewünschten Anwendungsfälle aus dem Lastenheft zu unterstützen.

Der konzeptuelle Entwurf der GUI besteht aus Bilders eines GUI-Builders oder aus Zeichnungen der geplanten Oberfläche der einzelnen Anwendungsfälle.

Auf dem Treffen zum MS3 spielen die Teams die Aktionen der Anwendungsfälle mit Hilfe des konzeptuellen Entwurfs der GUI in einer Präsentation für die Betreuer durch. Hierbei lassen sich Lücken und Unklarheiten leicht erkennen.

Das folgende Kapitel liefert Hinweise zur Gestaltung der GUI.

8.4. GUI der Anwendungsfälle

8.4.1. Robustheit

Im Code der GUI sollen falsche Eingaben und fehlende Angaben abgefangen werden und nicht zu einem Absturz oder einer Fehlfunktion (z.B. Zugriff auf leere Liste) führen.

8.4.2. Rückmeldung an den Benutzer

Die GUI soll dem Benutzer eine Rückmeldung nach der Durchführung einer vom Benutzer gewählten Funktion geben, z.B.: Nach dem Auswählen des Buttons zur Speicherung eines neuen Kunden in der Datenbank, soll die GUI eine Nachricht über das Ergebnis der Speicherung ausgeben, d.h. ob die Speicherung erfolgreich war oder nicht.

8.4.3. Komfort bei der Bedienung

Die GUI soll dem Benutzer ein Mindestma"s an Komfort bieten. Beispiel: Wenn die GUI dem Benutzer eine Liste von Kunden zur Auswahl anbietet, dann soll die getroffene Auswahl auch automatisch in vorhandene Eingabefelder eingefügt werden (z.B. der Name des Kunden) und der Benutzer soll diese Eingabe nach der Auswahl nicht noch manuell vornehmen müssen.

8.4.4. Beschriftungen der GUI-Elemente

Die Beschriftungen der GUI-Elemente (Reiter, Buttons, Eingabefelder, Ausgabefelder usw.) sollen für den Benutzer aussagekräftig und informativ sein und sollen passend zum realisierten Anwendungsfall gewählt werden. Beispiel: der Name des Buttons zum Anlegen eines neuen Kunden sollte nicht nur "Ok" sondern "Kunde anlegen" heißen.

8.4.5. Anordnung der GUI-Elemente

Die einzelnen Elemente der GUI (Buttons, Listen, Eingabefenster, usw.) sollen derart angeordnet sein, dass die korrekte Bedienung für den Benutzer leicht zu erkennen ist. Oftmals ist es besser, pro Anwendungsfall eine eigene Seite einzurichten, anstatt durch die gleiche Seite mehrere Anwendungsfälle zu unterstützen und dadurch die Zuordnung der Buttons zu den einzelnen Anwendungsfällen zu verlieren.

9. Hinweise zu MS4

Bis zum MS4 ist zu machen:

- Implementierung der verteilten Arbeitspakete in den entsprechenden Komponenten
  - Das Dokument Arbeitspakete MS4 gibt an, welcher Teilnehmer welche Anwendungsfälle (und ggfs. auch andere Aufgaben) implementieren soll.
- Entwicklerdokumentation der erstellten Steuerungsklassen in der Fachlogik-Schicht mit JavaDoc.
- Klassendiagramm anpassen
- Korrektur aller Bugs und Setzen des korrekten Status

Verbindlicher Termin, konkrete Liefergegenstände und genaue Abgabeorte sind in den Folien zur Einführungsveranstaltung angegeben.

Bei der Implementierung der GUI müssen natürlich auch die Hinweise aus Abschnitt 8.4 beachtet werden.

9.1. Vorgaben zur Implementierung

9.1.1. Versionsangaben in pom.xml

Die gegebenen Projekte in Git besitzen bereits eine von uns vordefinierte pom.xml. Die in dieser pom.xml angegebenen Versionen dürfen nicht verändert werden.

9.1.2. Qualität des Codes

Im Software-Praktikum sollen Sie nicht nur funktionierenden Code erstellen, sondern der erstellte Code soll auch möglichst professionell erstellt werden. Wir achten also bei der Überprüfung des Codes nicht nur darauf, dass der Code funktional korrekt ist, sondern auch darauf, dass er sauber, verständlich, durchdacht, effizient erstellt wurde.

Beispiel: Bei der Implementierung der Methoden der Datenschicht muss der Aufrufer a) den EntityManager setzen und b) oftmals (jedoch nicht immer)ein Objekt als Parameter übergeben. Es soll im Code der Datenschicht unbedingt überprüft werden, ob der EntityManager gesetzt wurde und ob als Parameter übergebene Objekte korrekt instanziiert sind. Das heißt, es sind die entsprechenden Überprüfungen im Code vorzunehmen (`em == null` bzw. `objekt == null`).

9.1.3. Verwendung der Gateway-Klasse für `EntityManager`

Zur Kapselung der Klasse `EntityManager` der `JPA` ist eine Gateway-Klasse für jedes System von uns bereits implementiert worden:

- BKS: die Klasse `IDatabaseImpl`
- GDS: die Klasse `IDatabaseImpl`
- WAWI: die Klasse `IDatabaseImpl`

Diese Klasse soll von allen Komponenten für den Zugriff auf die Datenbank verwendet werden. Die Verwendung dieser Gateway-Klasse erleichtert den Übergang von der Entwicklungs-Datenbank auf die Produktiv-Datenbank nach MS4 erheblich, da dann nur in dieser Klasse der Name der neuen `PersistenceUnit` geändert werden muss.

Diese Gateway-Klasse verwendet das Entwurfsmuster Singleton, um immer auf den gleichen `EntityManager` zuzugreifen und somit Inkonsistenzen beim Zugriff auf die Datenbank zu vermeiden. Das korrekte Objekt des `EntityManager` erhält man durch den Aufruf der Operation `getEntityManager()`.

9.1.4. Vorgegebene Komponenten-Schnittstellen

Die Schnittstellen der Komponenten und die zugehörigen Implementierungsklassen sind in den Dokumenten zum Grobentwurf definiert. Diese vorgegebenen Schnittstellen dürfen nicht verändert werden, d.h. es dürfen keine zusätzlichen Methoden erstellt werden.

9.1.5. Verwendung der Schnittstellen der Datenhaltung

Die Schnittstellen der Datenhaltung sind in den Dokumenten zum Grobentwurf definiert. Sie können von anderen Komponenten aus verwendet werden. In der Implementierung der Schnittstellen-Methoden werden oftmals Operationen auf Entitätsobjekten und der Datenbank durchgeführt. Es ist sehr wichtig, dass hierzu der richtige `EntityManager` verwendet wird.

Die Schnittstellen der Datenhaltung sollen in einer bestimmten Art und Weise verwendet werden:

- Die aufrufende Komponente setzt den zu verwendenden `EntityManager` mit der Methode `setEntityManager()`.
- Die aufrufende Komponente ist für die Transaktionsverwaltung zuständig (nicht die aufgerufene Komponenten-Schnittstelle).

Das Sequenz-Diagramm in Abbildung 1 illustriert den vorgegebenen Ablauf zur Verwendung einer Komponenten-Schnittstelle:
Der Klient, d.h. die Operation, die die Schnittstelle verwendet,

1. besorgt sich die korrekte Instanz des `EntityManager` von `IDatabaseImpl` (siehe Abschnitt 9.1.3),
2. übergibt diese an die Schnittstelle (`setEntityManager()`),
3. startet die Transaktion (`getTransaction().begin()`),
4. verwendet die gewünschten Operationen der Schnittstelle,
5. beendet die Transaktion (`getTransaction().commit()`).

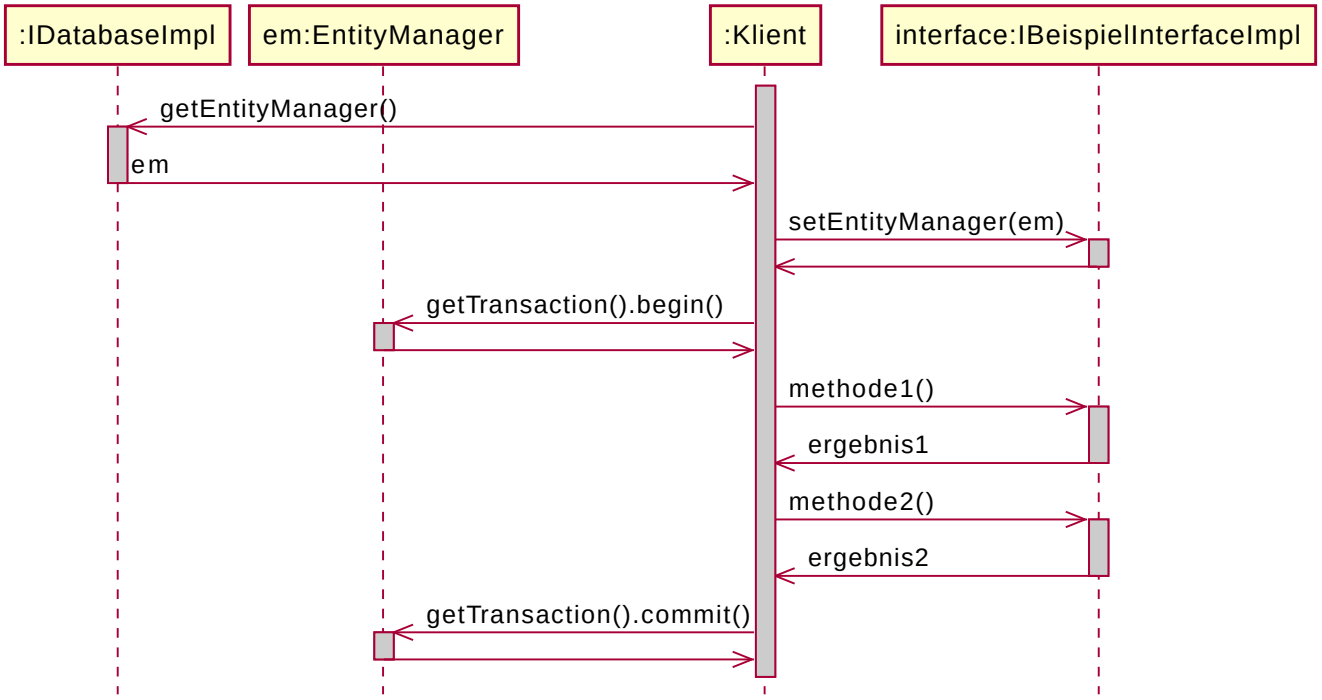

Abbildung 1. Sequenz-Diagramm zur Verwendung einer Komponenenten-Schnittstelle

Es ist also immer der Klient, d.h. der Aufrufer einer Operation der Komponenten-Schnittstelle der Datenhaltung, für das Setzen des `EntityManager` und die Transaktionsverwaltung verantwortlich!

9.2. Entwicklerdokumentation

Die Entwicklerdokumentation erfolgt in javadoc. Es müssen nur die Steuerungs-Klassen in den Komponenten der Schicht "Fachlogik" mit javadoc-Kommentare versehen werden - Die javadoc-Kommentare sollen nicht bei den Interface-Klassen, sondern nur bei den Implementierungen der Interface-Klassen erstellt werden.

Die javadoc-Kommentare sollen aussagekräftig sein. Sie sollen beschreiben, was eine Methode genau macht, welche Parameter erwartet werden und welches Ergebnis in welcher Situation geliefert wird.

10. Hinweise zu MS5

Bis zum MS5 ist zu machen:

- Korrektur aller Bugs und Setzen des korrekten Status.

Verbindlicher Termin, konkrete Liefergegenstände und genaue Abgabeorte sind in den Folien zur Einführungsveranstaltung angegeben.

Auf der Sitzung zum MS5 sollen alle Teams, die gemeinsam an einem System gearbeitet haben, die Lauffähigkeit dieses Systems vorführen. Hierzu werden die Schritte aus dem Integrationstest duchgeführt.