Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Prof. Dr. Sascha Alda
Moritz Balg, Dr. Rita Cornely

16. Oktober 2024

Übungen zur Vorlesung Software Engineering I
WS 2024 / 2025

Übungsblatt Nr. 3
(Abgabe Mittwoch, den 23. Oktober 2024, **09:00 Uhr**)

Aufgabe 1 (Modellierung Sprint Planning Meeting 1, 10 Punkte):

In der Vorlesung wurde der Ablauf des Scrum-Prozesses in Auszügen bereits modelliert (ggf. vorarbeiten). Es soll nun ihre Aufgabe sein, für ein Unternehmen insbesondere die Vorgehensweise bei der Durchführung des Planning Meetings Nr. 1 (vgl. Kapitel 2) durch ein UML-basiertes Aktivitätsdiagramm näher zu verdeutlichen. Man führt viele Aktivitäten hier zwar aus, es fehlt aber eine Gesamtübersicht über die Vorgehensweise. Der Geschäftsführer gibt dabei folgendes Statement ab:

„Nachdem der Product Owner (PO) die Agenda des Meetings und im Anschluss das Sprintziel vorgestellt hat, gibt er uns dann erst mal einen Überblick über die Anforderungen, die er in dem Sprint haben möchte. Das ist gut, um einen ersten Eindruck zu bekommen! Dann wird jede Anforderung *nacheinander einzeln* durchgegangen, im Team analysiert sowie Rahmenbedingungen und Einflussfaktoren (z.B. auf andere Anforderungen) exakt besprochen. Nach der Schätzung einer Anforderung durch die Methode „Planning Poker“ erfolgt dann die Entscheidung, ob die betreffende Anforderung in das Selektierte Produkt Backlog aufzunehmen ist oder nicht. Falls nein, wird sie mit der neuen Schätzung in das Produkt Backlog zurückgelegt. Nachdem alle Anforderungen besprochen wurden, folgt das „Forecast“ aller Beteiligten.“

Der Scrum Master des Unternehmens hat noch weitere Infos, die er ihnen gerne mitgeben möchte: „Halt! Bevor wir mit der Präsentation der Anforderungen beginnen, schauen wir noch mal auf die Ergebnisse des vorherigen Sprints und ermitteln dann die neue Kapazität (die Velocity) für den kommenden Sprint. Und das Ziel ist für mich zu Beginn nicht fix gesetzt! Falls wir eine Anforderung nach einer Entscheidung nicht in das Selektierte Produkt Backlog reinkommt, sollten wir im Anschluss stets überprüfen, ob dadurch das Ziel ggf. anzupassen ist! Zum Ende dokumentiere ich alle Ergebnisse (Gemeinsames Ziel, Selektiertes Produkt Backlog nebst Schätzungen“) im Tool „Confluence“, bevor wir dann in das 2. Planning Meeting eingehen, das ist aber ein separater Prozess, der für diese Aufgabe keine Rolle spielt.“

Berücksichtigen Sie in ihrem Modell auch die Dokumente „Ziel“ und „Selektiertes Produkt Backlog“. Swim Lanes brauchen sie *keine* zu berücksichtigen.

Aufgabe 2 (Weiterentwicklung Container mit diversen Mustern. 20 Punkte)

In dieser Aufgabe soll die Klasse `Container` aus dem Übungsblatt 2 weiterentwickelt werden. Falls sie diese Aufgabe noch nicht gelöst haben, dann können sie auf die Musterlösung der Übung 2 zurückgreifen (ab Mittwochabend online verfügbar). Als interne Datenstruktur soll eine Liste verwendet werden, die mit einem Typ `Member` als Generic parametrisiert ist. Folgende Änderungen (engl.: *change request* (*CR*)) sind dabei zu berücksichtigen:

CR1: Es soll zur Laufzeit zugesichert werden, dass von der Klasse `Container` nur *ein einziges Mal ein Objekt* erzeugt werden kann und somit nur ein Objekt davon im Speicher existiert. Wie lässt sich das realisieren, welches Pattern ist hier relevant? Hinweis: beachten sie in ihrem Lösungsweg die Verwendung einer *statischen* Methode zur Erzeugung des `Container`-Objekts. Erweitern sie die Klasse `Container` entsprechend. Weitere Hinweise dazu in der Übung.

CR2: Die Klasse `Container` soll um eine Methode `store` erweitert werden, welche die aktuell in einem `Container`-Objekt hinzugefügten `Member`-Objekte *persistent* auf einen Datenspeicher abspeichert. Die Methode hat folgende Signatur:

```
+ store() : void {throws PersistenceException}
```

Auch das Laden von `Member`-Objekten z.B. nach einem Neustart soll mit der Methode `load` möglich sein. Diese neue Methode der Klasse `Container` hat die Signatur:

```
+ load() : void {throws PersistenceException}
```

Befinden sich zum Zeitpunkt des Ladens `Member`-Objekte in der Liste, so sollen diese einfach überschrieben werden.

Von außerhalb eines Containers muss eine zugehörige Persistenz-Strategie in den Container flexibel *gesetzt* werden können. Eine Persistenz-Strategie muss das Interface `PersistenceStrategy` implementieren und die dort vorgegebenen Methoden überschreiben. Zudem muss über das Generic der Type definiert werden, der hier abgespeichert werden soll (hier: `Member`). Das Interface sowie die dazugehörigen Klassen (u.a. die Klasse `PersistenceException)` können sie aus dem GitHub-Repository der Vorlesung beziehen:

https://github.com/aldaGit/codesSEws24

Dort finden sie auch zwei Implementierungen des Interface `PersistenceStrategy` im Package `org.hbrs.se.ws24.exercises.uebung3.persistence`: die Klasse `Persistence-StrategyStream` sowie die Klasse `PersistenceStrategyMongoDB`.

Die Implementierung für MongoDB wird *noch* nicht unterstützt, ein Aufruf der Methoden wirft eine Exception. Eine Implementierung wird hier nicht erwartet von ihnen. Jedoch sollen sie das mögliche Werfen der Exception mit einem JUnit5-Test testen (siehe *CR4*).

Die Implementierung für Stream sollten sie in dieser Aufgabe hinzufügen. Zum Speichern muss die Klasse `ObjectOutputStream` aus dem Package `java.io` verwendet werden (bitte eigenständig recherchieren!). Zum Laden muss die Klasse Klasse `ObjectInputStream` aus dem Package `java.io` verwendet werden (bitte eigenständig recherchieren!). Einige Hinweise und einen guten Link (!) finden Sie in den Kommentaren im Source Code auf GitHub.

CR3: Refaktorisieren sie die Klasse `Container`, indem sie Methode `dump` zur Ausgabe von Member-Objekten aus der Klasse `Container` in eine neue Klasse `MemberView` bewegen (engl. *move*). Eine Ausgabe soll nur noch in der `MemberView`-Klasse über diese „bewegte“ Methode erfolgen, die von der `MemberView`-Klasse bereitgestellt wird mit einer neuen Signatur:

```
+ dump( liste : List<Member> )
```

Um die aktuelle Liste von Member-Objekten zu erhalten, sollte eine entsprechende Methode im `Container` bereitgestellt werden, bitte diese auch implementieren:

```
+ getCurrentList() : List<Member>
```

Geben sie auch eine Implementierung einer `Client`-Klasse an, welche diverse Member-Objekte erzeugt, in den Container hinzufügt, diese wieder mittels der Methode `getCurrentList` ausliest und dann in die Klasse `MemberView` zur Ausgabe übergibt. Achtung: Aus dieser `Client`-Klasse und auch aus der `MemberView`-Klasse darf keine Persistenz-Strategie in den Container gesetzt werden! Hinweis: Realisieren sie eine `Main`-Klasse, welche die notwendigen Instanziierungen und Initialisierungen durchführt.

Frage: Welches Pattern zur Implementierung und Bereitstellung der Persistenz-Strategien wird hier angewandt? Hinweise finden sich auch im Source Code ;-)

CR4: Entwickeln sie einen JUnit5-Testfall mit mindestens 4 aussagekräftigen Test-Methoden, um ihre Klasse Container *hinreichend* zu testen. Folgende Testfälle sollten dabei zumindest getestet werden (jeder Testfall eine Test-Methode):

- Keine Strategie von außen gesetzt („Test auf NULL“)
- Verwendung der Strategie „`PersistenceStrategyMongoDB`“
- Fehlerhafte Location des Files, in dem Streams ihre Objekte abspeichern (Tipp: FileStreams mögen keine Directories ;-))
- „Round-Trip-Test“: Objekt hinzufügen, Liste persistent abspeichern, Objekt aus Container löschen und Liste wieder einladen. Nach jedem Schritt den Zustand des Containers testen!