Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Prof. Dr. Andreas Hackelöer

25. April 2024

Übungen zur Vorlesung Software Engineering I
Sommersemester 2024

Übungsblatt Nr. 3

(Abgabe in Teams von max. 3 Personen bis: Donnerstag, den 2. Mai 2024, **9:00 Uhr**)

Wichtige Anmerkungen:

- Bitte reichen Sie sämtliche Abgaben nur noch über LEA ein, auch den Quellcode! Sie können dafür aus Ihrem GitLab- oder GitHub-Repository, sofern Sie eines verwenden, die Funktion „Download ZIP“ verwenden und das ZIP anschließend bei LEA einstellen.

- Bitte als Kompressionsformat ausschließlich ZIP verwenden, nicht RAR oder tar.gz!

- Abbildungen / UML-Diagramme idealerweise als PDF oder wenigstens als PNG oder JPG einreichen, bitte jedoch nicht in einem (XML-)Rohformat eines UML-Editors! Falls das Werkzeug Ihrer Wahl keine Bild-Exportfunktion bietet, können Sie dafür auch einfach die Taste PrintScrn/Drucken verwenden und den Screenshot anschließend in einem beliebigen Grafikprogramm bearbeiten.

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

Der Chef eines Unternehmens bittet Sie um eine Übersicht des Ablaufs eines Sprint Planning Meetings 1 mittels eines UML-Aktivitätsdiagramms. Er beschreibt den Ablauf wie folgt:

„Der Product Owner (PO) teilt Kaffee an alle Kaffeetrinker aus. Gleichzeitig teilt er Tee an alle Teetrinker aus. Wenn er mit beidem fertig ist, stellt er das Ziel des Sprints vor. Im Anschluss gibt er einen Überblick über alle Anforderungen des Sprints. Danach wird jede Anforderung hintereinander einzeln im Team diskutiert im Hinblick auf Rahmenbedingungen und Einflussfaktoren. Ist das geschehen, schätzt das Team den Aufwand für die Anforderung zusammen ab und entscheidet dann, ob die Anforderung in das Selektierte Product Backlog aufgenommen werden soll. Wenn ja, wird die Anforderung mit einer Priorität versehen und dann in das Selektierte Product Backlog aufgenommen. Wenn nein, wird sie mit der neuen Abschätzung in das Product Backlog zurückgelegt. Ist letzteres der Fall, überprüft das Team, ob das Sprintziel angepasst werden muss.

Sind alle Anforderungen bearbeitet, wird die neue Kapazität (Velocity) des Teams für den kommenden Sprint auf Basis der vorherigen Sprints bestimmt. Schließlich nimmt das Team den Forecast vor, also die Prognose, welche Backlog-Items in dem Sprint umgesetzt werden können. Am Ende werden alle Ergebnisse des Meetings, also Sprintziel und Selektiertes Product Backlog einschließlich der Schätzungen, im Tool Confluence dokumentiert."

Sie sollten in Ihrem Modell die Dokumente „Sprintziel" und „Selektiertes Product Backlog" mit abbilden. Swim Lanes müssen nicht berücksichtigt werden.

Aufgabe 2 (Weiterentwicklung CardBox mit Mustern, 20 Punkte)

Die Klasse `CardBox` aus dem Übungsblatt 2 soll weiterentwickelt werden. Falls Sie diese Aufgabe nicht gelöst haben, können Sie auf die Musterlösung zurückgreifen.

Folgende Änderungen (Change Requests, CR) sollen umgesetzt werden:

CR1:
Die Klasse `CardBox` soll nur ein einziges Mal im Speicher existieren. Wenden Sie ein dafür geeignetes Muster (Pattern) an und benennen Sie es auch. Verwenden Sie eine statische Methode zur Erzeugung Ihres `CardBox`-Objekts.

CR2:
Die Klasse `CardBox` soll um eine Methode `save()` ergänzt werden, welche die aktuell vorhandenen `PersonCard`-Objekte dauerhaft abspeichert, d.h. auf einem Speicher ablegt. Diese Methode hat diese Signatur:

```
+ save() : void {throws CardboxStorageException}
```

Auch soll sie um eine Methode `load()` ergänzt werden, welche die abgelegten Objekte wieder einliest. Diese Methode hat folgende Signatur:

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

Verwenden Sie die Java-Klassenbibliothek, um alle `PersonCard`-Objekte der `CardBox` in einer Datei zu speichern und von dort wieder zu laden. Befinden sich zum Zeitpunkt des Ladens bereits `PersonCard`-Objekte in der `CardBox`, sollen sie vorher gelöscht werden. Zum Speichern **muss** die Klasse `ObjectOutputStream` verwendet werden (dazu bitte eigenständig recherchieren!). Zum Laden **muss** die Klasse `ObjectInputStream` verwendet werden. Beide Klassen finden Sie in dem Package `java.io`.

Die Klasse `CardboxStorageException` muss von Ihnen spezifiziert und implementiert werden und soll im Fehlerfall Informationen darüber enthalten, warum das Speichern bzw. Laden nicht funktioniert hat.

CR3:
Refakturieren Sie Ihre Klasse `CardBox`, indem Sie die Methode `showContent()` zur Ausgabe von `PersonCard`-Objekten in eine neue Klasse namens `PersonCardView` verschieben. Die Ausgabe des Inhalts der `CardBox` soll nur noch in der `PersonCardView`-Klasse erfolgen. Dabei soll die Signatur von `showContent()` wie folgt angepasst werden:

```
+ showContent( liste : List<PersonCard> )
```

Um die aktuelle Liste von `PersonCard`-Objekten zu erhalten, sollte eine entsprechende Methode in der `CardBox` vorhanden sein, die ebenfalls zu implementieren ist:

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

Implementieren Sie außerdem eine `Client`-Klasse, welche mehrere `PersonCard`-Objekte erzeugt, diese der `CardBox` hinzufügt, sie alle mit der Methode `getCurrentList()` ausliest und dann der Klasse `PersonCardView` zur Ausgabe übergibt.

CR4:
Entwickeln Sie einen JUnit-Testfall, in dem die Funktionalität aus CR2 hinreichend getestet wird.