Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

06. Juni 2024

Übungen zur Vorlesung Software Engineering I
Sommersemester 2024

Übungsblatt Nr. 7

(Abgabe in Teams von max. 3 Personen bis: Mittwoch, den 19. Juni 2024, **9:00 Uhr**)

Achtung: Diese Übung ist umfangreicher, dafür gibt es jedoch eine Bearbeitungszeit bis zum 19.6.! Die Übungen in der nächsten Woche finden wie gewohnt statt, anstelle einer Vorstellung der Musterlösung können Fragen zu Vorlesungsinhalten gestellt werden.

Aufgabe 1 (Entwicklung UML-Modelle, 15 Punkte):

Bitte beachten Sie das Dokument „**Fallstudie NullPointer Collaboration Portal**“. Darin sind Anforderungen beschrieben für die folgenden Aufgaben:

a)
Entwerfen Sie ein (einziges!) UML Use Case-Modell aus dem gegebenen textuellen Use Case sowie den User Stories. (Achtung: Das Use-Case-Modell ist nicht sehr groß.) Dabei müssen die Use Cases aus den Vorbedingungen des textuellen Use Case nicht berücksichtigt werden.

b)
Wenden Sie auf den textuellen Use Case die Abbott-Methode an und entwickeln Sie ein Klassendiagramm zur Darstellung der Analyse-Objekte. Annotieren Sie jede Klasse mit einem Objekttyp. Modellieren Sie die wichtigsten Beziehungen zwischen den Klassen, ausgehend von der zentralen Control-Klasse „Jobangebot suchen“. Berücksichtigen Sie nur die DTOs, die sich aus der Spezifikation des textuellen Use Case direkt ergeben. Zwischen den Entities sollten die Kardinalitäten berücksichtigt werden (siehe auch die User Stories).

Aufgabe 2 (Pattern zur Weitergabe von Reports, UML; 15 Punkte)

Ein Anwendungsfall „Sende Reports“ soll entworfen werden. Konkret sollen zwei Formen von Report-Objekten verschickt werden können: `SkillReport`-Objekte und `CompanyReport`-Objekte. Für jeden `Report`-Typ soll es jeweils eine eigene konkrete `Publisher`-Klasse (Beispiel: `SkillReportPublisher`) geben. Alle `Report`-Objekte können durch eine von dem `Publisher` angebotene Methode `produce()` erzeugt werden (Beispiel: `SkillReportPublisher` erzeugt mit `produce()` Objekte vom Typ `SkillReport`).

`Publisher` sollen die Möglichkeit haben, `Consumer`-Objekte zu benachrichtigen, wenn ein Report erzeugt wurde. Es gibt für jeden `Report`-Typ ebenfalls einen eigenen `Consumer`-Typ (Beispiel: `SkillConsumer`). Hierzu führt jedes `Publisher`-Objekt eine Liste von registrierten `Consumer`n (Subscribern). Bei einem `Publisher` kann über die Methoden `register()` und `deregister()` ein Consumer der Liste der zu benachrichtigenden Consumer hinzugefügt bzw. von dieser entfernt werden. Außerdem besitzen Publisher eine Methode, um einen erzeugten `Report` mit einer bestimmten ID abzurufen (`getReport()`).

`Publisher` können bei Erzeugung eines `Report`s jeden `Consumer` in ihrer Subscriber-Liste benachrichtigen, indem sie die Methode `update()` jedes `Consumer`s aufrufen, was in der Methode `notify()` implementiert wird. Dabei erfährt der Consumer über die Übergabe eines Strings vom Thema (topic) des Reports; außerdem erfährt er die ID des erzeugten Reports (siehe nächsten Absatz).

Wird bei einem `Publisher` die Erzeugung eines Reports mit `produce()` aufgerufen, speichert der `Publisher` den neu erzeugten Report in einer internen Liste der erzeugten Reports (und benachrichtigt, wie oben beschrieben, die registrierten `Consumer`). Der `Publisher` vergibt eine ID für den neu erzeugten Report und speichert den Report unter dieser ID in seiner internen Liste.

`Consumer`-Objekte können, sobald sie benachrichtigt werden, entscheiden, ob das übergebene Thema (topic) für sie relevant ist. Dazu vergleichen sie das übergebene Thema mit einer Liste relevanter Themen (in Form von Strings). Beim `SkillConsumer` ist dies eine String-Liste namens „skills“. Wenn ein `Consumer`-Objekt den Report für relevant hält, kann es ihn über eine geeignete Methode `getReport()`, die der `Publisher` anbietet, von diesem beziehen.

Ein `SkillReport` hat ein Attribut `skill` vom Typ String sowie eine `reportID` (int) und einen `studentName` (String).

Consumer-Objekte können durch eine externe `Main`-Klasse bei den Publisher-Objekten registriert oder auch de-registriert werden.

Ihre Aufgaben:

a)
Modellieren Sie das aus der obigen Beschreibung entstehende UML-Klassendiagramm. Berücksichtigen Sie Methoden und eventuelle Attribute der Klassen und modellieren Sie die Signaturen der Methoden exakt. Sie brauchen keine Objekt-Typen zu annotieren. Die Klassen `CompanyReport` und `CompanyConsumer` müssen nicht weiter spezifiziert werden, sollten aber als Klassen im Diagramm enthalten sein. Beachten Sie bei der Modellierung nötige Generalisierungen durch abstrakte Klassen und/oder Interfaces.

b)

Modellieren Sie die Interaktionen der genannten Objekte als konkretes Szenario (mit Beispieldaten) in Form eines UML-Sequenzdiagramms. Verwenden Sie hierzu die synchrone Interaktion. Akteure müssen nicht modelliert werden. In diesem Sequenzdiagramm sollen drei verschiedene Interaktionen nacheinander stattfinden:

1. Erzeugung eines SkillConsumer-Objekts durch die Main-Klasse und Registrierung desselben bei einem SkillPublisher-Objekt. Gehen Sie dazu davon aus, dass der Publisher schon existiert und nicht neu erzeugt werden muss.

2. Erzeugung eines SkillReport-Objekts einschließlich aller daraus resultierenden Interaktionen. Die Erzeugung des SkillReport-Objekts wird beim Publisher mit produce() ausgelöst, dieser erzeugt daraufhin den Report und benachrichtigt das registrierte SkillConsumer-Objekt, welches wiederum sich den betreffenden Report vom Publisher holt (gehen Sie davon aus, dass das Topic relevant ist). Geben Sie bei der Erzeugung des SkillReport-Objekts die Attributwerte beispielhaft im Konstrukturaufruf (`new`(..)) an. Das interne Abspeichern des erzeugten Reports im Publisher brauchen Sie nicht modellieren.

3. Die De-Registrierung des erzeugten Consumer-Objekts durch die Main-Klasse beim Publisher.

c)
Modellieren Sie ein UML-Package-Diagramm. Dabei soll das Consumer-Interface mitsamt der es implementierenden Klassen in einem separaten „UI“-Package liegen. Alle anderen Klassen sollen in einem „AL“-Package liegen. Die Packages sollen die Klassen bzw. Interfaces explizit aufführen. Berücksichtigen Sie auch elementare <<import>>-Beziehungen zwischen den Packages.

Hinweis:
Das hier verwendete Muster zur Benachrichtigung von Objekten wird als Observer-Pattern bezeichnet ([Gamma, 1995]) und im noch kommenden Kapitel 6 der Vorlesung SE-1 vorgestellt. Weitere Informationen dazu finden Sie auch in (Brügge und Dutoit, 2013), Anhang A.7.

Bitte speichern Sie alle Ergebnisse als PDF und laden Sie sie auf LEA hoch.