Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Prof. Dr. Sascha Alda
Robert Hartmann (Dipl.-Inform.)

30. November 2021

Übungen zur Vorlesung Software Engineering I
WS 2021 / 2022

Übungsblatt Nr. 7
(Abgabe bis: Mittwoch, den 8. Dezember 2021, **09:00 Uhr**)

Aufgabe 1 (Entwicklung UML Modelle, 15 Punkte):

Die Entwicklung eines UML Use Case Modells sowie eines UML Klassendiagramms wird auf jeden Fall ein zentraler Bestandteil der Software Engineering I Klausur sein! Daher werden sie in diesem Aufgabenblatt nochmals auf Basis einer textuellen Use Case Beschreibung sowie User Stories UML-Modelle ableiten. Diese Anforderungen sind im Dokument „Fallstudie Coll@HBRS, v1.3“ beschrieben (siehe LEA). Ihre Aufgaben:

a.)
Entwickeln sie aus dem gegebenen textuellen Use Case sowie aus den User Stories (S. 2, bitte beachten!) *ein* UML Use Case Modell. Zeichnen sie auch die Systemgrenze ein. Achtung: das Use Case Modell ist „nicht besonders groß“ ;-) Die Use Case aus der Vorbedingung des textuellen Use Cases brauchen sie nicht zu berücksichtigen. Beeinflussen die User Stories das Use Case Modell? Antwort bitte kurz im Modell andeuten z.B. über einen Kommentar-Block.

b.)
Wenden sie auf den textuellen Use Cases die Abbott-Methode an, und entwickeln sie ein Klassendiagramm zur Darstellung der relevanten Analyse-Objekte. Jede Klasse sollte mit einem Objekttyp annotiert werden. Modellieren sie auch die wichtigsten Beziehungen zwischen den Klassen. Modellieren sie ihre Beziehungen ausgehend von der zentralen Control-Klasse „Jobangebot suchen“. Berücksichtigen sie nur die Datentransferobjekte, die sich explizit aus der Spezifikation des textuellen Use Case ergeben. Zwischen den Entities bitte auch die Kardinalitäten berücksichtigen (bitte auch hier insbesondere die User Stories beachten).

Aufgabe 2 (Herleitung Observer Pattern, UML; 15 Punkte)

Ein Objekt der Klasse `DataManager` soll in der Lage sein, `UI`–Objekte über neu hinzugefügte `Data`–Objekte zu benachrichtigen. `UI`–Objekte sind Objekte, die ein gegebenes Interface `UI` implementieren. Das Interface `UI` gibt dabei eine Methode `update() : void` vor, welche die abgeleiteten Objekte implementieren müssen.

Das Hinzufügen von `Data`–Objekten zu einem `DataManager` erfolgt ausschließlich über ein separates Objekt der Klasse `Producer` mittels der Methode `addData(d : Data)`.

Werden `UI`-Objekte aus dem `DataManager` nach dem Hinzufügen mittels der Methode `update` benachrichtigt, so können `UI`-Objekte *bei Bedarf* das neu hinzugefügte `Data`-Objekt aus dem `DataManager`-Objekt beziehen.

`UI`-Objekte können sich zur Laufzeit im `DataManager` zur Benachrichtigung registrieren und, falls notwendig, später auch wieder de-registrieren. `UI`-Objekte werden intern in dem `DataManager` in einer Liste abgespeichert. Auch `Data`-Objekte werden intern in einer Liste in dem `DataManager` abgespeichert.

Ihre Aufgaben:
a.)
Modellieren sie das aus der obigen Beschreibung resultierende UML Klassen-Diagramm. Berücksichtigen sie auch Methoden und eventuelle Attribute der Klassen. Werden Methoden nebst Signaturen vorgegeben, so verwenden sie diese exakt. Andere Methoden, die sich aus der Problemstellung ggf. ergeben, sollten sie selber mit einer geeigneten Signatur spezifizieren. Objekt-Typen brauchen sie keine zu annotieren.

b.)
Modellieren sie die Interaktionen der oben beschriebenen Objekte als *konkretes* Szenario in Form *eines* UML Sequenz-Diagramms. Verwenden sie hierzu die *synchrone* Interaktion. Akteure brauchen sie keine zu modellieren. In diesem Sequenz-Diagramm sollen drei verschiedene Interaktionen hintereinander (sequentiell) verdeutlicht werden:

- Die eigenständige Registrierung eines UI-Objekts in einen DataManager
- Das Erzeugen (!) und Hinzufügen eines Data-Objekts nebst den anschließenden Interaktionen zur Benachrichtigung und zum Beziehen des Data-Objekts durch das UI-Objekt
- Die eigenständige De-Registrierung eines UI-Objekts aus einen DataManager

c.)
Modellieren sie ein UML-Package Diagramm. Gehen sie davon aus, dass das `UI`-Interface nebst Klasse, welche das Interface implementiert, in einem separaten „UI“-Package liegt. Alle anderen Klassen liegen in einem „AL-Package“. Die Packages sollten die Klasse bzw. Interfaces explizit aufführen. Berücksichtigen sie auch elementare <<import>>-Beziehungen *zwischen* den Packages.

Hinweis: Das hier verwendete Muster zur Benachrichtigung von Objekten über eine Zustandsänderung wird als Observer Pattern bezeichnet ([Gamma, 1995], vgl. Kapitel 6, SE-1). Dies ist ein grundlegendes Muster zur flexiblen Interaktion von Objekten sowie von Subsystemen, das in vielen Bereichen verwendet wird! Sie sollten sich, durch eine erste Recherche, mit den grundlegenden Aspekten des Musters vertraut machen! Als geeignete Quelle verweise ich auf (Brügge und Dutoit, 2013), Anhang A.7.