Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

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

4. Oktober 2021

Übungen zur Vorlesung Software Engineering I
WS 2021 / 2022

Übungsblatt Nr. 1
(Abgabe bis: Mittwoch, den 13. Oktober 2021, **09:00 Uhr**)

Aufgabe 1 (Wiederholung Java, Erstes Muster, Blackbox-Testing, 25 Punkte):

In dieser Vorlesung werden sie eine Reihe von neuen Methoden zur Entwicklung von komplexen Software-Systemen kennenlernen. Ein solides Grundwissen aus den Bereichen Objektorientierung sowie Programmierung in Java ist dabei eine unabdingbare Voraussetzung für das Erlernen sowie für das Verständnis dieser Methoden! Daher werden in den Übungen Kenntnisse aus diesen Gebieten als Vorbereitung zu den Kapiteln 6 (Entwurfsmuster) und 7 (Entwicklung mit JUnit) wiederholt bzw. intensiviert sowie erste Themengebiete daraus adressiert.

Es seien zwei Java-Klassen `Client` und `GermanTranslator` sowie das Interface `Translator` gegeben, welches sie auf dem GitHub-Repository der Vorlesung herunterladen können:

https://github.com/aldaGit/codesSE2021

Weitere Infos zur Verwendung eines GitHub-Repository erhalten sie in Kürze.

Ihr konkreten Aufgaben:

1.)
Es soll ihre Aufgabe sein, eine Kommunikationsverbindung zu programmieren, so dass ein Objekt der Klasse `Client` zur Laufzeit die Methode `translateNumber( int zahl )` *direkt* von einem Objekt der Klasse `GermanTranslator` aufrufen kann. Dieser Aufruf sollte in dem konkreten Beispiel aus der Methode `display()` erfolgen. Die Bedingung ist dabei, dass in der Klasse `Client` *keine* Instanz von der Klasse `GermanTranslator` oder auch von einer anderen Klasse mit dem Befehl **new** unmittelbar erzeugt werden darf! Zudem soll in der Klasse `Client` *streng* gegen das Interface `Translator` implementiert werden.

- Wie kann diese Kommunikationsverbindung nun dennoch mit Hilfe einer *zusätzlichen* Klasse, welche die dazu notwendige Objekt-Erzeugung übernimmt, aufgebaut werden? In welchem Package sollte diese zusätzliche Klasse liegen?

- Welches Entwurfsmuster (engl.: *design pattern*) könnte für die Problematik der Objekt-Erzeugung verwendet werden? Was ist der software-technische Nutzen bei der Verwendung des Entwurfsmusters? Gratistipp: Hinweise für das korrekte Pattern finden sie bei unten angegeben Video-Tutorien ;-)

- Wie muss man den Source Code des Interface ggf. anpassen, um mögliche Kompilierfehler zu beseitigen?

2.)
Geben sie der Methode `translateNumber( int zahl )` eine Implementierung gemäß folgender Spezifikation:

- Für die *gültigen* Zahlen 1 bis 10 soll die schriftliche Schreibweise zurückgeliefert werden. Beispiel: 1 → eins; 2 → zwei .... usw.

- Falls eine *ungültige* Zahl größer 10 oder kleiner 1 übergeben wird, dann soll eine entsprechende Fehlernachricht („Übersetzung der Zahl [*übergebeneZahl*] nicht möglich ([*Ausgabe der Versionsnummer des Translators aus Interface*])“) als Rückgabewert der Methode zurückgegeben werden.

Für die Implementierung des Übersetzers dürfen sie keine Schleifen-Konstrukte (z.B. for, while etc.), *keine* Switch- oder if-Anweisung sowie keinen Bedingungsoperator / „Elvis-Operator“ (…? … : …) verwenden! Tipp: Denken sie über eine geeignete Datenstruktur zur Übersetzung nach!

3.)
Testen sie ihre Implementierung der Methode `translateNumer` aus der Klasse `GermanTranslator` mit einem Blackbox-Test. Über diese Test-Art sollten sie sich selber informieren, siehe dazu auch die Quelle (Spillner und Linz, 2012) auf LEA, Ordner Übung Nr. 1. Die Tests müssen in einer *separaten* (Test-)Klasse innerhalb eines eigenen Test-Packages platziert werden. Entwickeln sie eine repräsentative Menge an Positiv- und Negativtestfällen auf Basis geeigneter Äquivalenzklassen, um die Klasse *hinreichend* zu testen. Für die Implementierung des Blackbox-Tests können sie das Framework JUnit4 oder JUnit5 gerne verwenden. Zur Spezifikation der Testfälle und der Äquivalenzklassen sollten sie die Excel-Vorlage „TemplateTestCase v1.5“ / Blatt „Simple Test Suite“ verwenden (siehe Ordner Übung Nr. 1).

- Was ist der Vorteil einer separaten Test-Klasse?

- Was ist bei einem Blackbox-Test der Sinn von Äquivalenzklassen?

- Warum ist ein Blackbox-Test mit JUnit auf der Klasse `Client` nicht unmittelbar durchführbar?

Für all diese Teilaufgaben sollten sie die Source Codes als Lösung bereitstellen. Selbstverständlich können sich die Sources Codes aller Teilaufgaben in einem Projekt befinden, das zusammen gezippt auf LEA oder aber auf einem GitHub-Repository bereitgestellt werden kann. Bei der Bereitstellung auf einem GitHub-Repository bitte den HTTPS-Link dazu kopieren und auf LEA in einer Text-Datei hochladen (weitere Hinweise in den u.g. Video-Tutorien). Die hier gestellten Fragen sollten sie kurz in einer separaten Text-Datei beantworten, bitte daraus ein PDF erzeugen und auf LEA hochladen. Das Excel direkt hochladen oder auch hier eine PDF-Datei generieren aus dem Blatt.

Empfehlenswerte Literatur für den Black Box Test:
Spillner, A. und Linz, T.: Basiswissen für den Softwaretest. dpunkt.Verlag, 5. Auflage. 2012 (Kapitel 5, siehe Kopie auf LEA, Ordner Übung Nr. 1)

Aufgabe 1-2 (Modellierung mit mit UML-Klassendiagrammen, 5 Punkte)

Modellieren sie ihr finales Software-System aus der Aufgabe 1-1 (inkl. der Test-Klasse) als *einfaches* UML Klassendiagramm! Für diese, aber auch für die nächsten Übungen rund um die Modellierung von Software empfehle ich ihnen folgende Tools, die sie sich im Rahmen der ersten Woche anschauen und vergleichen sollten und daraus ihr präferiertes Tool zu entnehmen:

- UMLet 14.3

Gutes und übersichtliches Tool zur Modellierung, installierbar für Windows / MAC / Linux. Unterstützt keine Sequenzdiagramme (Anpassung möglich). Bedienung etwas gewöhnungsbedürftig zu Beginn, dann aber sehr effektiv ;-)
Quelle für Download: http://www.umlet.com/

- Draw.io

Schlankes browser-basiertes Tool, keine Installation auf ihrem Rechner notwendig! Abspeicherung der Dokumente in verschiedenen Formen möglich (Lokal, Cloud). Läuft nativ ohne Plugin auf allen gängigen Browsern.
Quelle: https://www.draw.io/

Mögliche Beziehungen zwischen den Klassen können sie in dieser Übung in Form einer einfachen Verwendungsbeziehung modellieren. Ziel soll es zudem sein, sich schon mal mit den Tools vertraut zu machen. Als ersten Einstieg in die Welt der Sprache UML empfehle ich folgendes Werk:

Seidl, Martina et al.: UML @ Classroom – An Introduction to Object-Oriented Modelling. Springer, 2015. (in Englisch, online verfügbar in der Bibliothek).

Eine gute Einführung zu Klassendiagrammen finden sie in diesem Buch in Abschnitt 4. Bitte das Modell in ein PDF-Dokument umwandeln und auf LEA hochladen.

Weitere Hinweise für die anstehenden Übungen:

Verwendung der Entwicklungsumgebung (IDE) IntelliJ und GitHub

In den Vorlesungen SE-1 (BWI) und SE-2 (BWI) wird die Verwendung der Entwicklungsumgebung IntelliJ empfohlen, die IDE hat sich in den letzten Jahren als Standard etabliert. Hier empfiehlt sich der Download der Ultimate-Version, die als registrierter Student kostenlos bezogen werden kann! Unübertroffen ist die Auto Completion Funktion, welche IntelliJ recht populär und beliebt gemacht hat. In den Übungen wird die IntelliJ für Demo-Zwecke verwendet. Auch die Verwendung eines GitHub-Repository ist hier sehr intuitiv realisiert worden.

https://www.jetbrains.com/idea/

Als Einführung in IntelliJ finden sie auf YouTube ein von mir produziertes mehrteiliges Video-Tutorial, welches die Installation des Tools, das Aufsetzen eines Projekts mit JUnit5 sowie die Integration von Git und GitHub verdeutlicht. Hier die Links:

Teil 1: Installation IntelliJ und Entwicklung eines Java-Projekts mit JUnit5 – https://www.youtube.com/watch?v=TNtRpkdW64s

Teil 2: Clone eines GitHub-Repository mit IntelliJ – https://www.youtube.com/watch?v=5nr4c3pwu3g

Teil 3: Push von Source Code auf ein GitHub-Repository mit IntelliJ https://www.youtube.com/watch?v=PbGiYUR9q0A

Die Links finden sie auch in LEA (Reiter: Nützliche Videos für Übungen). Hinweis: in diesen Videos wird auf ein Repository codesSE2020 verwiesen. Dieses Jahr verwenden wir natürlich das Repository codesSE2021.

Generelle Hinweise zu den Übungsaufgaben im Software Engineering:

Es kann in dieser Übung (und auch in anderen Übungen im Software Engineering) durchaus sein, dass es aufgrund komplexer Aufgabenstellungen bei einzelnen Teilaufgaben unterschiedliche Lösungen existieren, die bei der Bewertung bzw. bei der Besprechung als korrekt angenommen werden können. Die Musterlösungen repräsentieren dabei stets *eine mögliche* Variante, von der es durchaus Abweichungen in ihren Lösungen geben kann. Somit haben sie z.B. bei der Benennung von Variablen, Methodennamen oder generellen Entwurfsentscheidungen etc. *gewisse* Freiheitsgrade.