Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

11. April 2024

Übungen zur Vorlesung Software Engineering I
Sommersemester 2024

Übungsblatt Nr. 1

(Abgabe in Teams von max. 3 Personen bis: Mittwoch, den 17. April 2024, **9:00 Uhr**)

Aufgabe 1.1 (Wiederholung Java, Erstes Muster, Blackbox-Testing, 25 Punkte):

Diese Vorlesung behandelt Methoden zur Entwicklung komplexer Software-Systeme. Zur Vorbereitung werden in dieser Übung Grundlagen der objektorientierten Programmierung in Java wiederholt und vertieft. In den Kapiteln zu Entwurfsmustern und Softwaretesten werden diese Vorkenntnisse vorausgesetzt.

Gegeben sind die Klasse `Client` sowie das Interface `NumberTransformer`, herunterladbar auf dem GitLab-Repository der Vorlesung:

https://gitlab.com/hackeloeer/codesSS2024

Ihre Aufgaben:

Aufgabe 1.1a)

Implementieren Sie zwei verschiedene Varianten einer Implementierung des Interfaces `NumberTransformer`:

- Eine Variante `RomanNumberTransformer`, die beim Aufruf von `transformNumber()` die übergebene Zahl als römische Zahl zurückliefert. Diese Variante soll beim Aufruf von `getTransformerType()` den String „`Transformer für römische Zahlen`“ zurückliefern. Eine Beispielimplementierung der Konversion in römische Zahlen wird hier gegeben:

```
String[] romanSymbols = {"I", "IV", "V", "IX", "X", "XL", "L", "XC",
"C", "CD", "D", "CM", "M"};
int[] romanValues = {1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900,
1000};
int number = 1234;  // z.B.
StringBuilder romanNumeral = new StringBuilder();
for (int i = romanValues.length - 1; i >= 0; i--) {
    while (number >= romanValues[i]) {
        romanNumeral.append(romanSymbols[i]);
        number -= romanValues[i];
    }
}
String result = romanNumeral.toString();
```

- Eine Variante `GermanFormatNumberTransformer`, die beim Aufruf von `transformNumber()` die übergebene Zahl in deutscher Zahlenformatierung (Beispiel: „1425“ wird zu „1.425“) unter Verwendung der Java-Framework-Klasse `DecimalFormat` zurückliefert. Diese Variante soll beim Aufruf von `getTransformerType()` den String „`Transformer für deutsche Zahlenformatierungen`“ zurückliefern.

Beide Implementierungen sollen nur Zahlen zwischen 1 und 3000 akzeptieren und bei Aufrufen außerhalb dieses Wertebereichs die gleiche Fehlermeldung als String zurückgeben.

- **Frage 1:** Wie können Sie unter Berücksichtigung der Prinzipien des objektorientierten Entwurfs dafür sorgen, dass der Code, der den beiden Implementierungen gemeinsam ist, nicht dupliziert wird?

Aufgabe 1.1b)
Implementieren Sie die Klasse Client so, dass eine der Methode `printTransformation()` übergebene Zahl in die korrespondierende römische Zahl umgewandelt und diese ausgegeben wird, wobei Ihre Implementierung des `RomanNumberTransformer` aus Aufgabe 1.1 genutzt wird. Die Implementierung darf **keine** Instanz der Klasse `RomanNumberTransformer` oder einer anderen Klasse mit dem Schlüsselwort **new** unmittelbar erzeugen!

- **Frage 2:** Wie kann die Objekterzeugung mit Hilfe einer zusätzlichen Klasse durchgeführt werden? In welchem Package sollte diese zusätzliche Klasse liegen?

- **Frage 3:** Welches Entwurfsmuster liegt für diesen Anwendungsfall nahe? Welchen Vorteil bringt die Nutzung dieses Entwurfsmusters?

Aufgabe 1.1c)
Erstellen Sie einen Blackbox-Test für Ihre Implementierung der Methode `transformNumber()` der Klasse `RomanNumberTransformer`. Zur Vorgehensweise beachten Sie bitte die bereitgestellte Quelle (Spillner und Linz, 2012) auf LEA im Ordner zur Übung Nr. 1. Sie können sich grob an der bereitgestellten Klasse `TestRomanNumberTransformer` orientieren.

Die Tests müssen sich in einer separaten Test-Klasse in einem eigenen Test-Package befinden. Bestimmen Sie die auftretenden Äquivalenzklassen, identifizieren Sie jeweils einen geeigneten Repräsentanten und implementieren Sie Positiv- und Negativtestfälle, welche eine *hinreichende* Testabdeckung erzeugen. Es wird geraten, hierzu das Framework JUnit5 zu verwenden. Die Abhängigkeiten für JUnit 5 kann IntelliJ IDEA automatisch herunterladen und im Projektverzeichnis ablegen, wenn eine Testklasse mit JUnit-Annotationen angelegt wird.

Spezifizieren Sie die Testfälle und die Äquivalenzklassen in der Excel-Vorlage „TemplateTestCase“ (siehe LEA, Ordner zur Übung Nr.1).

- **Frage 4:** Warum sollten Testfälle in einer separaten Test-Klasse implementiert werden?

- **Frage 5:** Wozu dienen die Äquivalenzklassen im Blackbox-Test?

- **Frage 6:** Warum lässt sich für die Klasse `Client` nicht ohne weiteres ein Blackbox-Test umsetzen?

Allgemeine Hinweise zu dieser Aufgabe:

Bitte stellen Sie sämtlichen Quellcode mit der Lösung bereit. Natürlich können alle Teilaufgaben im gleichen Projekt bearbeitet sein. Sie können den Quellcode gezippt auf LEA hochladen oder auch in einem öffentlichen GitHub- oder GitLab-Repository bereitstellen und den HTTPS-Link dazu in einer Textdatei auf LEA hochladen.

Sämtliche hier gestellten Fragen sollten Sie in einer separaten Textdatei beantworten, daraus ein PDF erstellen und dieses auf LEA hochladen. Alternativ können Sie die Fragen auch in einer readme.md auf dem GitHub/GitLab-Repository beantworten. Aus dem ausgefüllten Excel-Sheet erstellen Sie bitte eine PDF-Datei, die Sie ebenfalls auf LEA hochladen.

Literaturempfehlung 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 UML-Klassendiagrammen, 5 Punkte):

Modellieren Sie nun Ihr Software-System aus der Aufgabe 1.2 (einschließlich der Test-Klasse) als UML-Klassendiagramm. Hierfür können Sie eines der folgenden Tools verwenden:

- Draw.io: https://app.diagrams.net/
  Browserbasiert, keine lokale Installation notwendig.

- UMLet 15.0: http://www.umlet.com/
  Modellierungstool für Windows / Macintosh / Linux. Unterstützt keine Sequenzdiagramme.

Beziehungen zwischen den Klassen können Sie für diese Übung in Form einer einfachen Verwendungsbeziehung modellieren. Bitte exportieren Sie das Modell in ein PDF-Dokument und laden Sie dieses auf LEA hoch.

Literaturempfehlung zu UML:
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.

Allgemeine Hinweise zu allen (auch den noch anstehenden) Übungen

Die Verwendung der Entwicklungsumgebung (IDE) JetBrains IntelliJ IDEA (https://www.jetbrains.com/idea) wird dringend empfohlen. Diese stellt einen Quasi-Standard in der Industrie dar. Registrierte Studierende können die Ultimate-Version kostenlos beziehen.

Falls nach der Installation ein importiertes Projekt nicht als Java-Projekt angezeigt wird, markieren Sie den Ordner oberhalb der Namespace-Hierarchie im Projekt-Explorer (Baumstruktur auf der linken Seite) und wählen im Kontextmenü „Mark Directory as -> Sources Root“ aus. JUnit-Testklassen sollten in einem separaten Namensraum abgelegt werden, der ebenfalls mit „Mark Directory as -> Test Sources Root“ markiert werden muss.

Bei Problemen mit der Bedienung von IntelliJ oder der Einrichtung des Projekts bitte eine Frage im Forum auf LEA stellen.

Für die meisten Aufgaben gibt es mehrere Lösungswege. Die Musterlösung stellt eine Variante davon vor, aber Lösungen mit anderen geeigneten Lösungswegen werden natürlich auch als korrekt bewertet.