Lösungen zur Übung 1.1 (©Fabius Schmitz und Tim Schulz)

Aufgabe 1.1)

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

- Um die Kommunikationsverbindung gemäß der Vorgabe zu implementieren, haben wir uns für eine Hilfsklasse ("Communication") entschieden. Diese wurde in das Paket Control eingefügt. Das Paket erschien uns als passend, da unsere Hilfsklasse von dort Zugriff auf die Objekte der anderen sich im Paket befindenden Klassen erstellen und deren Methoden nutzen kann. Des Weiteren haben wir so alles betreffend dem „Hintergrundablauf" im Package Control gebündelt.
- Zusatz: Man kann auch mit einer Factory Methode namens TranslateFactory arbeiten, wenn man in Zukunft noch andere Translator benutzen möchte. In der Aufgabenstellung als solche war dies unserer Ansicht nach nicht gefordert.

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?

- Als Design Pattern verwenden wir Erzeugungsmuster oder auch Factorypattern (Fabrikmuster) genannt. Wir entkoppeln die Erzeugung eines Objektes vom Rest des Codes. Der Software-technische Nutzen äußert sich dadurch, dass wir die Objekt-Erzeugung unabhängig von der tatsächlichen Implementierung realisieren können.

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

- Den Source Code des Interfaces haben wir nicht angefasst. Unsere Implementierung funktioniert dennoch für die von uns definierten Testfälle.

Aufgabe 1.2)

Für die Implementierung der Übersetzung gemäß den Vorgaben haben wir uns für die Verwendung eines String Arrays als Datenstruktur entschieden. Gemäß der als Parameter an die Methode übergebene Zahl wird der entsprechende Index als String zurückgegeben. Um die Sonderfälle (Zahl < 1 und Zahl > 10) haben wir eine ArrayIndexOutOfBoundsException in Kombination mit einem try-catch Block verwendet. So erhalten wir das geforderte Ergebnis.

Aufgabe 1.3)

Was ist der Vorteil einer separaten Testklasse?

- Wir kapseln den Test vom Rest unseres Programms. Dies führt zu einer erhöhten Übersichtlichkeit unserer Programmstruktur und es kommt nicht zu unerwünschten Wechselwirkungen zwischen Test und Rest des Programms.

-

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

- Bestimmte Testfälle lassen sich in Äquivalenzklassen einordnen (“gleiches Zusammenschmeißen”). Ist dies erfolgt, können wir die Äquivalenzklassen anhand eines Repräsentanten als ganzes prüfen anstatt alle Elemente der Äquivalenzklasse einzeln.

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

- Dies ist daher nicht möglich, weil die Methode “display” keinen Rückgabeparameter hat. JUnit braucht jedoch einen Rückgabewert für den Test.

Aufgabe Excel-Tabelle:

Simple Test Suite	
Test Object:	GermanTranslator

		Parameter	
TestCase No.	**Category** (pos; neg)	**input** (für weitere Input-Variablen können Spalten hinzugefügt werden)	**Output** (Erwartetes Ergebnis)
1	neg	-5	Fehler
2	neg	0	Fehler
3	pos	1	„eins“
4	pos	2	„zwei“
5	pos	3	„drei“
6	pos	4	„vier“
7	pos	5	„fuenf“
8	pos	6	„sechs“
9	pos	7	„sieben“
10	pos	8	„acht“
11	pos	9	„neun“
12	pos	10	„zehn“
13	neg	200	Fehler

Zugehörige Äquivalenzklassen:

Parameter	**Äquivalenzklasse**	**Repräsentant**	**Category** (pos; neg)
input	<=0	-5	neg
input	1-10	5	pos
input	>=11	200	neg