WS2023 SE1

Übungsblatt Nr. 1

Aufgabe 1

Frage: 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? Bitte beachten Sie dabei auch die Hinweise bzw. Anforderungen aus den Kommentaren der Klassen, Methoden und des Interfaces.

Antwort: Neue Klasse TranslatorFactory.java muss angelegt werden. In der neuen Klasse findet dann die Objekt-Erzeugung statt. Die Neue Klasse sollte sich in dem control package befinden(in dem selben package wie Client.java). Die Antworten auf die Fragen werden auch in dem Source Code deutlich.

Frage: 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 ;-)

Antwort: In unserem Fall verwenden wir die Factory-Methode als Design Pattern. Die Hauptgründe dafür sind:

1. Zentrale Steuerung der Objekt-Erzeugung. Es gibt eine Klasse in der wir das Erzeugen von Objekten verwalten und anpassen können. Dies erhöht auch die Übersichtlichkeit des Codes.
2. Entkopplung von dem Client-Code und den konkreten Klassen, die erstellt werden. Der Client muss nur die abstrakte Schnittstelle kennen und nicht die konkrete Implementierung.

3. Das Anpassen des Codes wird dadurch in Zukunft auch vereinfacht und übersichtlicher gehalten.
4. Bietet eine saubere, erweiterbare Lösung für die Objekterzeugung in der Softwareentwicklung.

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

Antwort: Das Interface musste public gemacht werden und die Methode translateNumber() muss static sein. Das sieht man auch im Source Code.

Aufgabe 2

Lösung in Source Code.

Aufgabe 3

Frage: Was ist der Vorteil einer separaten Test-Klasse?

Antwort: Eine eigene Test-Klasse isoliert den Testcode von dem Produktionscode. Somit wird die Wartung und die Lesbarkeit des Codes verbessert. Klare Trennung von Test- und Produktionscode. Am besten sogar in einem separaten Package.

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

Antwort: Ziel der Bildung von Äquivalenzklassen ist es, eine hohe Fehlerentdeckungsrate mit einer möglichst geringen Anzahl von Testfällen zu erreichen.

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

Antwort: Ein Blackbox-Test auf der Klasse Client ist nicht unmittelbar durchführbar, weil die Klasse Client eine Verbindung zu einer konkreten Implementierung, z.B. GermanTranslator, hat, die in der Regel nicht im Blackbox-Test berücksichtigt werden

sollte. Ein Blackbox-Test sollte auf der funktionalen Spezifikation und den Schnittstellen basieren.