Software Engineering - Übung 1

Gruppe:

- Max Grünewald
- Miguel Obrebski
- Deniz Ari

Link zum GitHub-Repo

Anmerkungen zur Abgabe

Aufgabe 1.1 a)

Die Grenzwerte waren nicht ganz eindeutig ("zwischen"), daher haben wir uns für das logischste entschieden und die Zahlen inklusive 1 und 3000 umzuwandeln.

Beantwortung der Fragen

1. Frage

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

Es gibt drei Möglichkeiten, um Duplikation zu vermeiden:

1. **Komposition**: Man platziert den gemeinsamen Code in einer separaten Klasse und verwendet diese Klasse in den beiden Implementierungen.
2. **Vererbung**: Ähnlich wie bei der Komposition, jedoch leitet man die spezifischen Implementierungen von dieser Klasse ab.
3. **Interfaces**: Man deklariert ein Interface mit den gemeinsamen Methoden und implementiert dieses Interface in den verschiedenen Implementierungen.

Wir haben dies in diesem Fall mit Vererbung gelöst.

2. Frage

Wie kann die Objekterzeugung mit Hilfe einer zusätzlichen Klasse durchgeführt werden? In welchem Package sollte diese zusätzliche Klasse liegen?

Die Objekterzeugung kann effizient durch das Entwurfsmuster der Factory-Klasse realisiert werden. Dies verbessert die Flexibilität und Wartbarkeit des Codes. In der Regel sollte die Factory-Klasse im gleichen Package wie die zu erzeugenden Objekte platziert werden, um den Zugriff auf eventuell paket-private/protected Konstruktoren oder Mitglieder zu ermöglichen.

Ein weiterer Ansatz wäre die Verwendung einer Kompositionsklasse, die vorab erzeugte, statische Objekte bereitstellt. Dies kann sinnvoll sein, wenn bestimmte Objektkonfigurationen häufig verwendet werden und deren Erzeugungskosten relativ hoch sind. Diese Kompositionsklassen sollten dort platziert werden, wo sie am häufigsten benötigt werden oder auch in speziellen, projektweiten Packages.

Wir haben uns der Einfachheit halber für eine Kompositionsklasse entschieden.

3. Frage

Welches Entwurfsmuster liegt für diesen Anwendungsfall nahe? Welchen Vorteil bringt die Nutzung dieses Entwurfsmusters?

Für diesen Anwendungsfall liegt ein Creational Pattern "Erstellungsmuster" nahe. Dieses Entwurfsmuster bietet die Möglichkeit, Objekte situationsgerecht und den Bedürfnissen des Systems entsprechend zu erstellen.

4. Frage

Warum sollten Testfälle in einer separaten Test-Klasse implementiert werden?

Durch die Implementierung von Testfällen in separaten Klassen wird der Produktionscode vom Testcode getrennt. Diese klare Trennung erleichtert das Codeverständnis und die Wartung.

5. Frage

Wozu dienen die Äquivalenzklassen im Blackbox-Test?

Äquivalenzklassen im Blackbox-Test dienen dazu, Eingabedaten in Gruppen zu unterteilen, um effizientere und effektivere Tests durchführen zu können. Die Grundidee ist, nur einen Vertreter aus jeder Klasse zu testen, da jede Äquivalenzklasse ähnliche Verhaltensweisen des Systems aufweist.

6. Frage

Warum lässt sich für die Klasse Client nicht ohne weiteres ein Blackbox-Test umsetzen? Die Klasse Client hat keine Ausgabewerte und wodurch Repräsentanten nicht einfach ermittelt/geprüft werden können.