# Übung 05

Korrektur: Sebastian Pritz

Name: Muegem Boral

## Punkte: 64

Allgemeine Anmerkungen:

## Beispiel 1: Flugreisen 21 von 30

Lösungsidee: 2/3

| Anmerkung | Abzug |
|---|---|
| Laut Erklärung werden die Flüge (oder Flugreisen? Geht nicht gut heraus) direkt beim Passagier/Person gespeichert? Das ist keine gute Idee aus mehreren Gründen:<br>1. Dann müsste man, um alle Passagiere für einen Flug zu erhalten über ALLE jemals gemeldeten Personen iterieren. Das ist ein enormer Aufwand in einem realen Szenario.<br>2. Manche Personen buchen eventuell mehrere Flüge gleichzeitig. Wenn nur ein/e Flug(reise) beim Passagier gespeichert ist, wäre das z.B. auch nicht abbildbar (dadurch dass bei dir die Person Passagier heißt ist das aber auch weniger sinnvoll)<br>➔ Personen sollten entweder bei der Flugreise oder beim Flug gespeichert werden (Flugnummer führt <u>direkt</u> zu allen betroffenen Passagieren/Infos) | Hinweis |
| << Operator nicht beschrieben | -1 |

Quellcode: 12/20

| Anmerkung | Abzug |
|---|---|
| Die Modellierung macht, wie oben bereits erklärt, leider wenig Sinn. | **-3** |
| Macht es Sinn, keine Informationen zu einer Person/Passagier zu haben? Man sollte in diesem Fall über Konstruktoren gewisse Anforderungen stellen | **Hinweis** |
| Verschiedene Klassen sollten auch in verschiedene Files aufgeteilt werden → Person – person.h/person.cpp, etc. | **-2** |
| Const nicht verwendet wenn möglich (set Funktionen z.B) | **-1** |
| TestNoPersonalInfo versucht Person nur mit PlaneTrip zu initialisieren, aber dafür gibt es keinen Konstruktor und es kompiliert nicht? | **-2** |

Testfälle: 7/7

| **Anmerkung** | **Abzug** |
| --- | --- |
| Durch die falsche Modellierung ergeben sich natürlich auch andere Testfälle, aber die gegebenen Testfälle passen zur gewählten Modellierung, deswegen kein weiterer Abzug. | |

## Beispiel 2: Stücklistenverwaltung 43 von 70

Lösungsidee: 0/7

| **Anmerkung** | **Abzug** |
|---|---|
| Die Lösungsidee beschreibt hier nicht das eigentliche Problem, sondern fasst nur kurz Vererbung und überschreiben zusammen, was nicht das Ziel war. Hier wäre gefordert gewesen, dass Part und Compositepart, sowie deren Zusammenhang beschrieben wird. Außerdem sollte auf den Formatter und die Load/Store Methoden eingegangen werden = Was ist zu implementieren/beachten | -7 |

Quellcode: 28/40

| **Anmerkung** | **Abzug** |
|---|---|
| Wie bei Beispiel 1: Bitte verschiedene Klassen aufteilen auf mehrere .h und .cpp Files! | **FF** |
| Der Konstruktor von CompositePart macht so wenig Sinn. Das Attribut „name“ in der Klasse Part, sollte als „protected“ definiert werden, damit auch CompositeParts darauf zugreifen können, und einen Namen besitzen, ohne direkt ein Part in die Liste zu speichern. | **-2** |
| Die Tests für die Formatter funktionieren nur für Parts, aber nicht für CompositeParts. Das Ziel sollte, wie in den Beispielen ersichtlich, sein, Parts UND CompositeParts strukturiert auszugeben, nicht einen Vektor von Parts (wobei CompositePart eh nur eine „verschachtelte Liste“ mit Namen ist). → Nachschauen ob (Down-)Cast auf CompositeParts möglich, und wenn ja, printParts mit Einrückung darauf aufrufen (rekursiv) | **-6** |
| Das Interface Storable sollte Parts und CompositeParts aus Files laden können und hat grundsätzlich nichts mit den Hierarchy/Set Formattern zu tun. Aus deinem Format kann man z.B. nicht Erkennen, falls etwas ein CompositePart ist | **-4** |
| Nun zur Beantwortung der Frage: Bei den Testfällen und einigen anderen Stellen im Code wurde auf das Schlüsselwort „new“ vergessen (also z.B. „Part p = Part(...)“ statt „Part* p = *new* Part(...)“). Die Objekte existieren also nur statisch in der angelegten Methode und können daher auch nicht mit „delete“ freigegeben werden. Deswegen musste auch händisch mit & auf Pointer konvertiert werden. Die Vorgehensweise mit delete parts[i] und anschließendem vector.clear() wäre sonst korrekt. Hoffe das beantwortet deine Frage. 😊 | **Hinweis** |

Testfälle: 15/23

| **Anmerkung** | **Abzug** |
|---|---|
| Beispiel aus der Angabe nicht vorhanden (bzw. zumindest ein komplexeres Beispiel) → Im HierarchyFormatter ist keine Hierarchie auf mehreren Ebenen sichtbar (Tisch enthält nur Parts, aber keine CompositeParts = „doppelt verschachtelt“) | **-2** |
| Equals und GetName beim CompositePart nicht getestet → Name nicht zuweisbar von außen (existiert private) = daher auch alle equal mit Name „ „ (leer) | **-4** |
| addPart mit einem CompositePart als Parameter nicht getestet | **-2** |