

## SWE3 -Uebung 5

Ausarbeitungsdauer: 10h

Datum: 26.12.2022

### Contents

Beispiel 1 – Flugreisen .....	1
Lösungsidee .....	1
Testfälle .....	2
Beispiel 2 – Stücklistenverwaltung .....	3
Lösungsidee .....	3
Testfälle .....	4

Disclaimer: Der Übungszettel konnte aufgrund von Krankheit und eines Familienurlaubes in den Weihnachtsferien nicht vollständig erledigt werden. Aus Mangel an Zeit wurden daher bis auf einzelne Testfälle, die Testfälle nur beschrieben, aber nicht ausimplementiert.

### Beispiel 1 – Flugreisen

#### Lösungsidee

Person	Flight	Flight_trip
<ul style="list-style-type: none"> <li>- first_name</li> <li>- last_name</li> <li>- gender</li> <li>- address</li> <li>- age</li> <li>- credit card</li> </ul> <ul style="list-style-type: none"> <li>+ get_first_name()</li> <li>+ get_last_name()</li> <li>+ get_gender()</li> <li>+ get_address()</li> <li>+ get_age()</li> <li>+ get_credit card()</li> </ul>	<ul style="list-style-type: none"> <li>- passenger_list;</li> <li>- number;</li> <li>- airline;</li> <li>- dep_airport;</li> <li>- arr_airport;</li> <li>- dep_time;</li> <li>- arr_time;</li> <li>- duration;</li> </ul> <ul style="list-style-type: none"> <li>+ add_passenger()</li> <li>+ get_passengers()</li> </ul>	<ul style="list-style-type: none"> <li>- flights_list</li> </ul> <ul style="list-style-type: none"> <li>+ add_flights()</li> <li>+ add_passenger()</li> <li>+ print_passenger_list()</li> <li>- print()</li> </ul>

Für jede Person werden Name, Adresse, Geschlecht, Alter und eine Kreditkarte hinterlegt. Die Adresse setzt sich aus Straße, PLZ, Stadt/Ort und Land zusammen. Es können einzelne Flüge angelegt werden. Dazu wird die Flugnummer, Fluglinie, Abflug und Ankunftsflughafen und Zeit und die Flugdauer benötigt. Informationen über einen Flug können nur bei der Erstellung übergeben, danach aber nicht mehr verändert werden. Es können Personen(Kunden) zu Flügen hinzugefügt werden und eine Liste der Passagiere ausgegeben werden. Eine Flugreise besteht aus einem Vektor von Flügen, welche die Hin- und Rückflüge

inkludieren. Die Flüge können entweder direkt bei der Erstellung der Flugreise übergeben werden, oder zu einem späteren Zeitpunkt, aber jedoch bevor der erste Kunde zur Reise hinzugefügt wird. Ebenfalls muss der User dafür sorgen, dass die Flüge in der richtigen Reihenfolge gespeichert werden. Wird ein Kunde der Flugreise hinzugefügt, so wird dieser zu allen Flügen hinzugefügt. Außerdem können alle Passagiere aller Flüge ausgegeben werden. Auch eine Übersicht aller Flüge der Flugreise können ausgegeben werden.

## Testfälle

Person:

- Alle Getter von Person – geben die jeweilige Information aus
- Konstruktor von Person

Flug:

- Konstruktor von Flug mit allen Attributen
- Konstruktor von Flug ohne Attribute - Error
- Add\_passenger mit einer Person
- Get\_passengers von Flug mit keiner Person
- Get\_passengers von Flug mit einer Person
- Get\_passengers von Flug mit mehreren Personen
- Flugdetails mit dem überladenen Operator << ausgeben

Flugreise:

- Defaultkonstruktor ohne Parameter
- Add\_flights nachträglich aufrufen
- Konstruktor mit flights vektor
- Konstruktor mit flights vektor und dann mit add\_flights überschreiben
- Add\_passenger zu Flugreise ohne Flüge – nichts passiert
- Add\_passenger zu Flugreise mit Flügen
- Print\_passenger\_list von Flügen ohne Passagiere
- Print\_passenger\_list von Flügen mit mehreren Passagieren
- Print\_passenger\_list mit immer dem gleichen Passagier
- Überladener operator << bei Flugreise ohne Flügen
- Überladener operator << bei Flugreise mit Flügen – details aller Flüge werden ausgegeben

Test constructor Flug und überladener operator << :

```
/Users/anjasmwab/CLionProjects/flugreisen/cmake-build-debug/flugreisen
AA938475   Air Blah   Duration: 1.5
Departure: Linz 7:00 02.02.2022
Arrival: Frankfurt 8:30 02.02.2022
```

Test constructor Flugreise mit flugvector und operator <<

```
Details of this trip:
AA938475   Air Blah   Duration: 1.5
Departure: Linz 7:00 02.02.2022
Arrival: Frankfurt 8:30 02.02.2022
AF9238475  Air Germany Duration: 6
Departure: Frankfurt 9:00 02.02.2022
Arrival: Denver 15:00 02.02.2022
ZR98475    Spirit    Duration: 3
Departure: Denver 17:00 02.02.2022
Arrival: Las Vegas 20:00 02.02.2022
UH3478475  Spirit    Duration: 3
Departure: Las Vegas 7:00 17.02.2022
Arrival: San Francisco 10:00 17.02.2022
OD98479895 Air Canada Duration: 10
Departure: San Francisco 10:00 17.02.2022
Arrival: München 20:00 17.02.2022
EB998345   Air Canada Duration: 2
Departure: München 21:00 17.02.2022
Arrival: Linz 23:00 17.02.2022
```

Test add\_passenger zu flug und print\_passenger\_list von einer Flugreise:

```
Print all passengers:
Anja Schwab
```

## Beispiel 2 – Stücklistenverwaltung

### Lösungsidee

Die Stücklistenverwaltung wird durch zwei vorgegebene Klassen realisiert. Part ist dabei die Basisklasse, von der die Klasse composite part alle Methoden und Attribute erbt. Ein composite part ist ein part aber ein part muss kein composite part sein. Composite part erbt das Namensattribut von part, welches daher dort protected gemacht werden muss. Bei beiden kann über den getter der Name ermittelt werden. Composite part hat zusätzlich einen Container, welcher partobjekte enthält. Diese können nun selbst wieder parts oder composite parts sein. Ein composite part kann also part von einem anderen composite part sein. Sollen also alle Teile ausgegeben werden, welche für ein Part benötigt/eingekauft werden müssen, müssen rekursiv alle Einzelteile der composite parts und dann wieder deren teile usw. ermittelt werden. Die Funktion equals in part soll mit virtual gekennzeichnet und dann in composite part überschrieben werden, da der Vergleich unterschiedlich ist, je nachdem ob zwei parts oder zwei composite parts miteinander verglichen werden. Der getter von name muss keine dynamisch gebundene Funktion sein, da dieser in allen abgeleiteten Klassen gleich ist, wie in der Basisklasse. In der Klasse composite part gibt es eine Methode um partobjekte zum jeweiligen composite part hinzuzufügen.

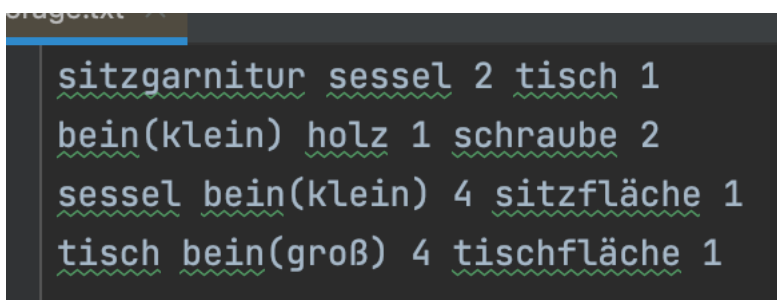
Ein leeres Part ohne Namen kann nicht existieren – daher kein defaultkonstruktor.

Zwei part objekte sind dann gleich, wenn sie den gleichen Namen besitzen. Zwei composite\_part objekte sind dann gleich, wenn sie den gleichen Namen besitzen und aus den gleichen Parts zusammengesetzt sind.

Die Klasse Formatter ist eine abstrakte Klasse: von ihr können also keine Instanzen erstellt werden. Sie enthält die pure virtual function printParts. Die von ihr abgeleiteten Klassen setformatter und hierachy formatter sind nicht abstrakt und implementieren jeweils die funktion print\_parts. Der Setformatter gibt eine Stückliste aller Einzelteile mit der jeweiligen Anzahl aus; dies kann wie eine Einkaufsliste zum Bau des composite parts verstanden werden. Der hierachy formatter hingegen gibt in einer hierachischen Struktur aus, welche Teile jeweils zu welchem anderen Teil gehören, also wie die Teile sich jeweils zusammensetzen.

Um die Stücklisten für später speichern und auch Stücklisten wieder laden zu können gibt es das Interface storable. Ein interface ist eine abstrakte Klasse, welche nur pure virtual functions bzw. nur Methodendeklarationen aber keine Methodendefinitionen enthält. Composite\_part erbt von storable und implementiert die Methoden aus.

Eine Textdatei muss wie folgt aufgebaut sein, um von load in ein composite\_part gelesen werden zu können:



```
sitzgarnitur sessel 2 tisch 1
bein(klein) holz 1 schraube 2
sessel bein(klein) 4 sitzfläche 1
tisch bein(groß) 4 tischfläche 1
```

In der ersten Zeile steht der Name des composite part selbst und die Teile aus denen er sich zusammensetzt. Danach folgen die restlichen Teile – immer die kleinste Einheit zuerst und erst danach die Teile die dieses kleinere Teil dann benötigen.

Store schreibt ein composite\_objekt in genau diesem Format in eine Datei.

Es wird davon ausgegangen, dass die zu ladenden Dateien das richtige Format besitzen.

## Testfälle

Part:

- Default konstruktor mit Name des parts
- Equals mit 2 gleichen parts
- Equals mit 2 unterschiedlichen parts (unterschiedlicher name)
- Getter des namens

Composite Part

- Konstruktor mit Name

- Konstruktor mit Name und Vektor von parts
- Equals mit 2 gleichen composite parts
- Equals mit 2 unterschiedlichen composite parts
- Equals mit 1 part und 1 composite part
- Add part mit einem part
- Add part mit einem composite part
- Get parts und dann partsvektor bearbeiten
- Load von einem nichtexistierenden File
- Load von einem existierenden File mit korrekter Struktur
- Load von einem File mit nur einem part
- Load von einem File mit mehreren parts
- Store von einem composite objekt ohne parts
- Store von einem composite objekt mit mehreren parts

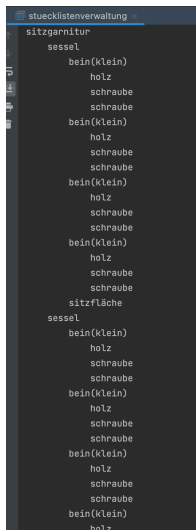
Formatter:

- Set\_formatter von einem part
- Set\_formatter von einem composite\_part
- Hierarchy formatter von einem part
- Hierarchy formatter von einem composite\_part

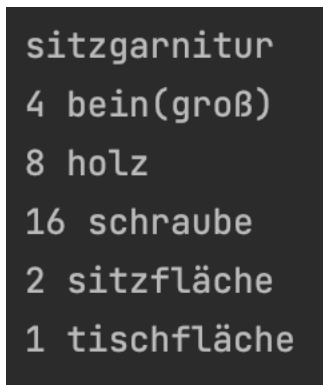
Test part constructor, composite\_part – name und vektor konstruktor und hierarchy und set formatter:

```
/Users/anjasmw/CLionProjects/stuecklistenverwaltung/cmake-build-debug/stuecklistenverwaltung
sitzgarnitur
  sessel
    bein (klein)
    bein (klein)
    bein (klein)
    bein (klein)
    sitzfläche
  sessel
    bein (klein)
    bein (klein)
    bein (klein)
    bein (klein)
    sitzfläche
  tisch
    bein (groß)
    bein (groß)
    bein (groß)
    bein (groß)
    tischfläche
sitzgarnitur
4 bein (groß)
8 bein (klein)
2 sitzfläche
1 tischfläche
```

Test load und hierarchy formatter:



Test load und set formatter:



Test store:

