

ÜBUNG 05

INHALT

Flugreisen	2
Lösungsidee	2
Class Person	2
Class Flight	2
Class Flight Travel	2
Testfälle	2
Testing Class Person	2
Testing Class Flight	3
Testing class FLight Travel	5
Stücklistenverwaltung	8
Lösungsidee	8
Partslists	8
Storable	8
Formatters	9
Testfälle	9
Testing PartsList	9
Testing Formatter	11
testing storable	14
Zeitaufwand	16

FLUGREISEN

LÖSUNGSDIEE

Es werden drei unabhängige Klassen benötigt, wobei ein Flug mehrere Personen enthalten kann und eine Flugreise jeweils mehrere hin- und Rückflüge enthält.

CLASS PERSON

Die Klasse besitzt alle benötigten Attribute für eine Person und die dazugehörigen get Methoden.

CLASS FLIGHT

Die Klasse Flug besitzt neben den vorgegebenen Attributen auch noch einen Vektor mit pointer auf Personen (mit einer dazugehörigen add- und get- Methode).

Für die Klasse wurde ein eigener Ausgabe Operator überlagert, um eine Flug und die dazugehörigen Personen einfach ausgeben zu können.

CLASS FLIGHT TRAVEL

Die Klasse flight_travel besitzt zwei Vektoren mit Pointer auf Flüge für die Hin- und Rückflüge.

Die Methode `static void print_flights(std::ostream& os, vector<flight*>& flights)` muss static sein da sonst nicht vom Operator << darauf zugegriffen werden kann.

TESTFÄLLE

TESTING CLASS PERSON

DEFAULT CONSTRUCTOR

Input: `person person;`

Output:

```
Default constructor
Person:
Name:
Gender:
Age: -1
Address:
Credit card number: -1
```

ASSIGN CONSTRUCTOR

Input: `person person("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);`

Output:

```
Assign constructor
Person:
Name: Max Mustermann
Gender: maennlich
Age: 20
Address: 1 Musterstrasse
Credit card number: 123
```

GET_FIRST_NAME()

Input: `person` `person("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);`
Output:

```
Firstname: Max
```

`GET_LAST_NAME()`

Input: `person` `person("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);`
Output:

```
Lastname: Mustermann
```

`GET_GENDER()`

Input: `person` `person("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);`
Output:

```
Gender: maennlich
```

`GET_AGE()`

Input: `person` `person("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);`
Output:

```
Age: 20
```

`GET_ADDRESS()`

Input: `person` `person("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);`
Output:

```
Address: 1 Musterstrasse
```

`GET_CREDIT_CARD_NR()`

Input: `person` `person("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);`
Output:

```
Credit card number: 123
```

TESTING CLASS FLIGHT

DEFAULT CONSTRUCTOR

Input: `flight` `flight;`
Output:

```
Testing class flight...
Default constructor
Flight:
Flight number: -1
Airline:
From-To: invalid start - invalid end
Time: 00:00 - 00:00
Flight duration (in h): -1
```

ASSIGN CONSTRUCTOR

Input: `flight` flight(123, "Eurowings", "Wien", "Paris", "Mon Aug 09:30 2017", "Mon Aug 11:30 2017", 2);

Output:

```
Assign constructor
Flight:
Flight number: 123
Airline: Eurowings
From-To: Paris - Wien
Time: Mon Aug 09:30 2017 - Mon Aug 11:30 2017
Flight duration (in h): 2
```

`GET_AIRLINE();`

Input: `flight` flight(123, "Eurowings", "Wien", "Paris", "Mon Aug 09:30 2017", "Mon Aug 11:30 2017", 2);

Output:

```
Airline: Eurowings
```

`GET_DEPARTURE_LOCATION();`

Input: `flight` flight(123, "Eurowings", "Wien", "Paris", "Mon Aug 09:30 2017", "Mon Aug 11:30 2017", 2);

Output:

```
Departure Location: Paris
```

`GET_ARRIVAL_LOCATION();`

Input: `flight` flight(123, "Eurowings", "Wien", "Paris", "Mon Aug 09:30 2017", "Mon Aug 11:30 2017", 2);

Output:

```
Arrival Location: Wien
```

`GET_DEPARTURE_TIME();`

Input: `flight` flight(123, "Eurowings", "Wien", "Paris", "Mon Aug 09:30 2017", "Mon Aug 11:30 2017", 2);

Output:

```
Departure Time: Mon Aug 09:30 2017
```

`GET_ARRIVAL_TIME();`

Input: `flight` flight(123, "Eurowings", "Wien", "Paris", "Mon Aug 09:30 2017", "Mon Aug 11:30 2017", 2);

Output:

```
Arrival Time: Mon Aug 11:30 2017
```

`GET_FLIGHT_DURATION();`

Input: `flight` `flight(123, "Eurowings", "Wien", "Paris", "Mon Aug 09:30 2017", "Mon Aug 11:30 2017", 2);`

Output:

```
Flight Duration: 2
```

`ADD_PERSONS();`

Input:

```
flight flight(123, "Eurowings", "Wien", "Paris", "Mon Aug 09:30 2017", "Mon Aug 11:30 2017", 2);
person p1("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);
person p2("Lisa", "Mueller", "weiblich", 24, "24 Schmidtstrass", 970);
person p3("Julia", "Schmidt", "weiblich", 30, "13 Muehlstrasse", 7499);
flight.add_person(&p1);
flight.add_person(&p2);
flight.add_person(&p3);
cout << &flight;
```

Output:

```
Test add person:
Eurowings Nr: 123 From Wien To Paris Duration: 2h
--- Passenger List ---
Max Mustermann
Lisa Mueller
Julia Schmidt
```

`GET_PERSONS();`

Input:

```
flight flight(123, "Eurowings", "Wien", "Paris", "Mon Aug 09:30 2017", "Mon Aug 11:30 2017", 2);
person p1("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);
person p2("Lisa", "Mueller", "weiblich", 24, "24 Schmidtstrass", 970);
person p3("Julia", "Schmidt", "weiblich", 30, "13 Muehlstrasse", 7499);
flight.add_person(&p1);
flight.add_person(&p2);
flight.add_person(&p3);
vector<person*> persons_;
persons_ = flight.get_persons();
cout << "Passengers:\n";
for (size_t i = 0; i < persons_.size(); i++)
{
    person* p = persons_[i];
    cout << p->get_first_name() << " " << p->get_last_name() << std::endl;
}
```

Output:

```
Test get persons:
Passengers:
Max Mustermann
Lisa Mueller
Julia Schmidt
```

TESTING CLASS FLIGHT TRAVEL

DEFAULT CONSTRUCTOR

Input:

```
flight_travel ft;  
cout << ft;
```

Output:

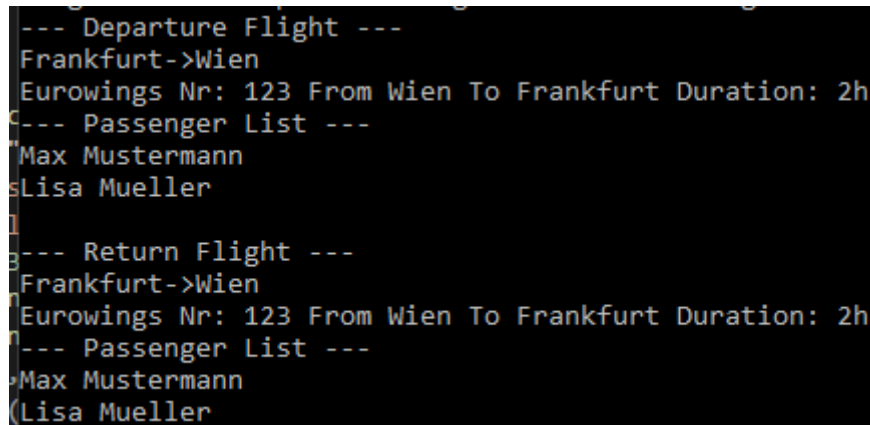
```
Flight has no departure flight -- Invalid flight !!!!
```

ASSIGN CONSTRUCTOR

Input:

```
person p("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);  
person p2("Lisa", "Mueller", "weiblich", 24, "24 Schmidtstrass", 970);  
person p3("Julia", "Schmidt", "weiblich", 30, "13 Muehlstrasse", 7499);  
flight fd1(123, "Eurowings", "Wien", "Frankfurt", "Mon Aug 09:30 2017", "Mon Aug  
10:30 2017", 2);  
fd1.add_person(&p);  
fd1.add_person(&p2);  
flight fr(456, "Austrian Airlines", "Paris", "Linz", "Sun Aug 05:00 2017", "Sun  
Aug 09:30 2017", 2.5);  
fr.add_person(&p);  
fr.add_person(&p2);  
fr.add_person(&p3);  
flight_travel ft(&fd1, &fr);  
cout << ft;
```

Output:



```
--- Departure Flight ---  
Frankfurt->Wien  
Eurowings Nr: 123 From Wien To Frankfurt Duration: 2h  
--- Passenger List ---  
Max Mustermann  
Lisa Mueller  
--- Return Flight ---  
Frankfurt->Wien  
Eurowings Nr: 123 From Wien To Frankfurt Duration: 2h  
--- Passenger List ---  
Max Mustermann  
(Lisa Mueller
```

GET_DEPARTURE_FLIGHT();

Input:

```
person p("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);  
person p2("Lisa", "Mueller", "weiblich", 24, "24 Schmidtstrass", 970);  
person p3("Julia", "Schmidt", "weiblich", 30, "13 Muehlstrasse", 7499);  
flight fd1(123, "Eurowings", "Wien", "Frankfurt", "Mon Aug 09:30 2017", "Mon Aug  
10:30 2017", 2);  
fd1.add_person(&p);  
fd1.add_person(&p2);  
flight fr(456, "Austrian Airlines", "Paris", "Linz", "Sun Aug 05:00 2017", "Sun  
Aug 09:30 2017", 2.5);  
fr.add_person(&p);  
fr.add_person(&p2);  
fr.add_person(&p3);  
flight_travel ft(&fd1, &fr);  
vector<flight*> departure_flights;  
departure_flights=ft.get_departure_flights();  
cout << "Departure flight:\n";
```

```

for (size_t i = 0; i < departure_flights.size(); i++)
{
    cout << departure_flights[i];
}

```

Output:

```

Departure flight:
Eurowings Nr: 123 From Wien To Frankfurt Duration: 2h
--- Passenger List ---
Max Mustermann
Lisa Mueller

```

```

GET_RETURN_FLIGHT();

```

Input:

```

person p("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);
person p2("Lisa", "Mueller", "weiblich", 24, "24 Schmidtstrass", 970);
person p3("Julia", "Schmidt", "weiblich", 30, "13 Muehlstrasse", 7499);
flight fd1(123, "Eurowings", "Wien", "Frankfurt", "Mon Aug 09:30 2017", "Mon Aug
10:30 2017", 2);
fd1.add_person(&p);
fd1.add_person(&p2);
flight fr(456, "Austrian Airlines", "Paris", "Linz", "Sun Aug 05:00 2017", "Sun
Aug 09:30 2017", 2.5);
fr.add_person(&p);
fr.add_person(&p2);
fr.add_person(&p3);
flight_travel ft(&fd1, &fr);
vector<flight*> return_flight;
return_flight = ft.get_return_flights();
cout << "Return flight:\n";
for (size_t i = 0; i < return_flight.size(); i++)
{
    cout << return_flight[i];
}

```

Output:

```

Return flight:
Austrian Airlines Nr: 456 From Paris To Linz Duration: 2.5h
--- Passenger List ---
Max Mustermann
Lisa Mueller
Julia Schmidt

```

```

OP();

```

Input:

```

person p("Max", "Mustermann", "maennlich", 20, "1 Musterstrasse", 123);
person p2("Lisa", "Mueller", "weiblich", 24, "24 Schmidtstrass", 970);
person p3("Julia", "Schmidt", "weiblich", 30, "13 Muehlstrasse", 7499);
flight fd1(123, "Eurowings", "Wien", "Frankfurt", "Mon Aug 09:30 2017", "Mon Aug
10:30 2017", 2);
fd1.add_person(&p);
fd1.add_person(&p2);
flight fr(456, "Austrian Airlines", "Paris", "Linz", "Sun Aug 05:00 2017", "Sun
Aug 09:30 2017", 2.5);
fr.add_person(&p);
fr.add_person(&p2);
fr.add_person(&p3);
flight_travel ft(&fd1, &fr);

```

```
cout << ft;
```

Output:

```
--- Departure Flight ---
Frankfurt->Wien
Eurowings Nr: 123 From Wien To Frankfurt Duration: 2h
--- Passenger List ---
Max Mustermann
Lisa Mueller

--- Return Flight ---
Frankfurt->Wien
Eurowings Nr: 123 From Wien To Frankfurt Duration: 2h
--- Passenger List ---
Max Mustermann
Lisa Mueller
```

STÜCKLISTENVERWALTUNG

LÖSUNGSIDEE

PARTSLISTS

Part ist die Basisklasse für **CompositePart** weil ein zusammengesetztes Teil besteht aus mehreren Teilen. Der Konstruktor von **CompositePart** ruft den Basiskonstruktor (von **Part**) auf und initialisiert somit das Attribut `name`.

Darüber hinaus wird hier das Interface (abstrakte Klasse) **Storable** angelegt und der Klasse **CompositePart** vererbt. **Storable** besitzt die zwei Methoden `store` und `load` welche ein Composite Part in ein file speichern oder es vom file laden. Die beiden Methoden sind pure virtual functions und werden nur in der Klasse **CompositePart** implementiert.

STORABLE

Anmerkung: Die Formatierung ist die gleiche wie beim Hierachy Formatter (selbe Funktion).

STORE:

Store speichert das gesamte Konstrukt formatiert in einem File.

Print:

Print speichert zuerst einen String („padding“) mit der Anzahl an Einrückungen nach Level. Solange ein Teil Unterteile hat, wird dann bei jedem Unterteil überprüft, ob dieses wieder Unterteile hat (also ein weiteres Level tiefer geht) oder nicht. Wenn ja, wird die Funktion rekursiv aufgerufen, wenn nicht, dann wird das padding + einen weiteren Tab (da wir uns ja immer das nächste Element und nicht das derzeitige Element anschauen) ausgegeben.

LOAD:

Die erste Zeile im File ist der Name des derzeitigen Objektes (z.b. Sitzgarnitur). Die anderen Daten im File werden in einem Tupel, dass jeweils das Level (um wie viele Tabs eingerückt) und den Namen des derzeitigen Teils speichert. Das derzeitige Level bekommt man durch das Zählen der Tabs in einer Line, wobei die Tabs vor dem Speichern im Tupel aus dem String gelöscht werden müssen.

Parse:

Parse wird von load aufgerufen und ist eine rekursive Funktion welche schließlich die gesamte Struktur von dem Teil erkennt, da wir die Hierarchie selbst noch nicht wissen. Es überprüft also ob das derzeitige Element schon das Ende dieser Hierarchie ist (z.b. Sitzfläche) , ob es ein Level tiefer geht (z.b. Bein) oder ob es sogar mehr Level tiefer geht. Wenn der erste Fall erreicht ist, returnt die Funktion das gesamte CompositeTeil.

FORMATTERS

Die Klasse **Formatter** ist eine abstrakte Klasse (runde Klammern) und die Basisklasse von **SetFormatter** und **HierarchyFormatter**. Weil es sich um eine abstrakte Klasse handelt, wird die Methode zur pure virtual function und zusätzlich gleich 0 gesetzt was bedeutet, dass die Methode nicht hier, sondern in den abgeleiteten Klassen implementiert wird. Da man eine pure virtual function hat, wird auch ein virtual destructor benötigt.

Wichtig ist, dass in jeder abgeleiteten Klasse die Methode implementiert wird, da die Klasse sonst selbst als abstrakte Klasse gesehen wird.

Override wird bei dieser Methode in den abgeleiteten Klassen nicht benötigt, da diese noch nicht implementiert wurde und damit nicht überschrieben werden muss.

HIERACHY FORMATTER

-> Selbe print Funktion wie schon oben beim Store
überprüft zusätzlich ob das übergebene Teil überhaupt valide ist.

SET FORMATTER

Der Set Formatter legt eine map mit den Namen eines Teils und die Anzahl wie oft dieses vorkommt an.

CountParts

CountParts überprüft ob diese Teil schon in der Map existiert. Wenn nicht wird es hinzugefügt, wenn schon dann wird der counter für dieses Teil um 1 erhöht.

TESTFÄLLE

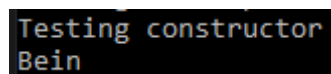
TESTING PARTSLIST

PART CONSTRUCTOR

Input:

```
Part p("Bein");  
std::cout << p.get_name() << std::endl;
```

Output:



```
Testing constructor  
Bein
```

GET_NAME()

Input:

```
Part bein_klein("Bein (klein)");  
std::string s = bein_klein.get_name();  
std::cout << s << std::endl;
```

Output:

```
Testing get_name()
Bein (klein)
```

EQUALS()

Input:

```
Part bein_1("Bein (klein)");
Part bein_2("Bein (klein)");
Part flaeche_1("Sitzflaeche");
Part flaeche_2("Tischflaeche");

std::cout << bein_1.get_name() << " and " << bein_2.get_name() << std::endl;
if (bein_1.equals(bein_2)) {
    std::cout << "Equal!" << std::endl;
}
else {
    std::cout << "Not Equal!" << std::endl;
}

std::cout << flaeche_1.get_name() << " and " << flaeche_2.get_name() <<
std::endl;
if (flaeche_1.equals(flaeche_2)) {
    std::cout << "Equal!" << std::endl;
}
else {
    std::cout << "Not Equal!" << std::endl;
}
}
```

Output:

```
Testing equals()
Bein (klein) and Bein (klein)
Equal!
Sitzflaeche and Tischflaeche
Not Equal!
```

COMPOSITEPART CONSTRUCTOR

Input:

```
CompositePart sessel("Sessel");
std::cout << sessel.get_name() << std::endl;
```

Output:

```
Not Equal!
Testing constructor
Sessel
```

GET_PARTS()

Input:

```
CompositePart sessel("Sessel");

Part bein_klein("Bein (klein)");
Part sitzfleache("Sitzflaeche");

sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
```

```

sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&sitzfleaech);

```

```

HierachyFormatter h_fmt;

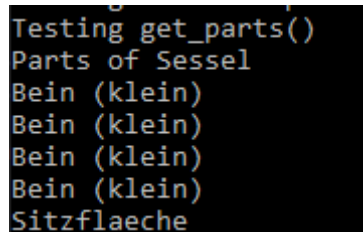
```

```

h_fmt.PrintParts(&sessel);

```

Output:



```

Testing get_parts()
Parts of Sessel
Bein (klein)
Bein (klein)
Bein (klein)
Bein (klein)
Sitzflaeche

```

ADDPARTS()

Input:

```

CompositePart sessel("Sessel");

```

```

Part bein_klein("Bein (klein)");
Part sitzfleaech("Sitzflaeche");

```

```

sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&sitzfleaech);

```

```

vector<Part*> parts;

```

```

parts = sessel.getParts();

```

```

std::cout << "Parts of " << sessel.get_name() << std::endl;

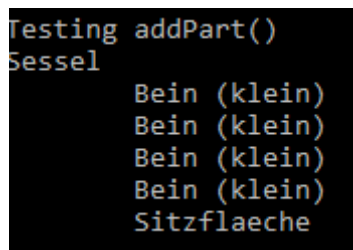
```

```

for (int i = 0; i < parts.size(); ++i) {
    std::cout << parts[i]->get_name() << std::endl;
}

```

Output:



```

Testing addPart()
Sessel
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Sitzflaeche

```

TESTING FORMATTER

SET_FORMATTER_INVALID_PART();

Input:

```

CompositePart sessel("Sessel");

```

```

SetFormatter set_fmt;

```

```
set_fmt.PrintParts(&sessel);
```

Output:

```
Testing set formatter with invalid part
Invalid part was given to formatter - has no sub parts
```

```
SET_FORMATTER_SESSEL();
```

Input:

```
CompositePart thing("Sitzgarnitur");

CompositePart sessel("Sessel");
Part bein_klein("Bein (klein)");
Part sitzfleache("Sitzflaeche");

sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&sitzfleache);

thing.addPart(&sessel);
thing.addPart(&sessel);
```

```
SetFormatter set_fmt;
```

```
set_fmt.PrintParts(&sessel);
```

Output:

```
Invalid part was given to formatter -
Testing set formatter with sessel
Sessel
    4 Bein (klein)
    1 Sitzflaeche
```

```
SET_FORMATTER_SITZGARNITUR();
```

Input:

```
CompositePart thing("Sitzgarnitur");
CompositePart sessel("Sessel");
CompositePart tisch("Tisch");
Part bein_klein("Bein (klein)");
Part bein_big("Bein (gross)");
Part sitzfleache("Sitzflaeche");
Part tischfleache("Tischflaeche");

sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&sitzfleache);

tisch.addPart(&bein_big);
tisch.addPart(&bein_big);
tisch.addPart(&bein_big);
tisch.addPart(&bein_big);
tisch.addPart(&tischfleache);
```

```

thing.addPart(&sessel);
thing.addPart(&sessel);
thing.addPart(&tisch);

SetFormatter set_fmt;

set_fmt.PrintParts(&thing);

```

Output:

```

Testing set formatter with Sitzgarnitur
Sitzgarnitur
    4 Bein (gross)
    8 Bein (klein)
    2 Sitzflaeche
    1 Tischflaeche

```

```

HIERACHY_FORMATTER_INVALID_PART();

```

Input:

```

CompositePart sessel("Sessel");
HierarchyFormatter h_fmt;

h_fmt.PrintParts(&sessel);

```

Output:

```

Testing hierachy formatter with invalid part
Invalid part was given to formatter - has no sub parts

```

```

HIERACHY_FORMATTER_SESSEL();

```

Input:

```

CompositePart thing("Sitzgarnitur");

CompositePart sessel("Sessel");
Part bein_klein("Bein (klein)");
Part sitzfleache("Sitzflaeche");

sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&sitzfleache);

thing.addPart(&sessel);
thing.addPart(&sessel);

HierarchyFormatter h_fmt;

h_fmt.PrintParts(&sessel);

```

Output:

```

Testing hierachy formatter with sessel
Sessel
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Sitzflaeche

```

```
HIERACHY_FORMATTER_SITZGARNITUR();
```

Input:

```
CompositePart thing("Sitzgarnitur");
    CompositePart sessel("Sessel");
    CompositePart tisch("Tisch");
    Part bein_klein("Bein (klein)");
    Part bein_big("Bein (gross)");
    Part sitzfleache("Sitzflaeche");
    Part tischfleache("Tischflaeche");

    sessel.addPart(&bein_klein);
    sessel.addPart(&bein_klein);
    sessel.addPart(&bein_klein);
    sessel.addPart(&bein_klein);
    sessel.addPart(&sitzfleache);

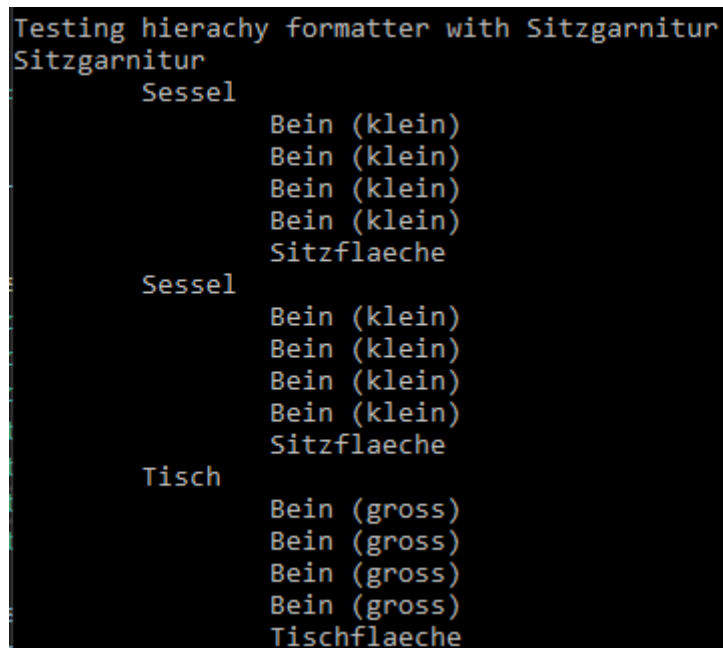
    tisch.addPart(&bein_big);
    tisch.addPart(&bein_big);
    tisch.addPart(&bein_big);
    tisch.addPart(&bein_big);
    tisch.addPart(&tischfleache);

    thing.addPart(&sessel);
    thing.addPart(&sessel);
    thing.addPart(&tisch);

    HierachyFormatter h_fmt;

    h_fmt.PrintParts(&thing);
```

Output:



```
Testing hierachy formatter with Sitzgarnitur
Sitzgarnitur
    Sessel
        Bein (klein)
        Bein (klein)
        Bein (klein)
        Bein (klein)
        Sitzflaeche
    Sessel
        Bein (klein)
        Bein (klein)
        Bein (klein)
        Bein (klein)
        Sitzflaeche
    Tisch
        Bein (gross)
        Bein (gross)
        Bein (gross)
        Bein (gross)
        Tischflaeche
```

TESTING STORABLE

```
LOAD();
```

Input:

```

CompositePart thing("Sitzgarnitur");
CompositePart sessel("Sessel");
CompositePart tisch("Tisch");
Part bein_klein("Bein (klein)");
Part bein_big("Bein (gross)");
Part sitzfleache("Sitzflaeche");
Part tischfleache("Tischflaeche");

sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&sitzfleache);

tisch.addPart(&bein_big);
tisch.addPart(&bein_big);
tisch.addPart(&bein_big);
tisch.addPart(&bein_big);
tisch.addPart(&tischfleache);

thing.addPart(&sessel);
thing.addPart(&sessel);
thing.addPart(&tisch);

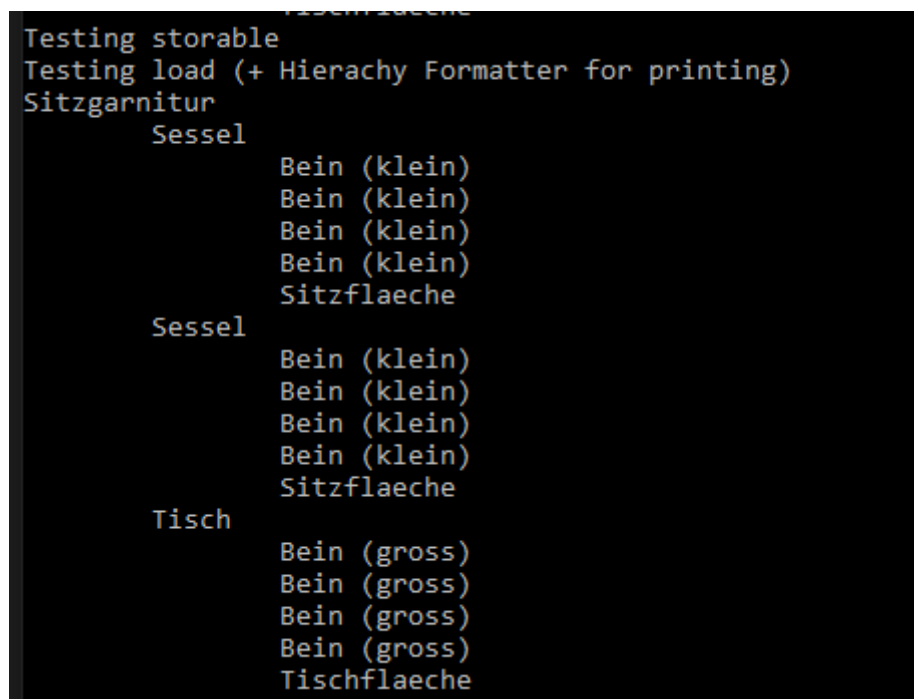
std::ifstream y("test_load.txt");
thing.load(y);
y.close();

HierachyFormatter h_fmt;

h_fmt.PrintParts(&thing);

```

Output:



```

Testing storable
Testing load (+ Hierachy Formatter for printing)
Sitzgarnitur
  Sessel
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Sitzflaeche
  Sessel
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Sitzflaeche
  Tisch
    Bein (gross)
    Bein (gross)
    Bein (gross)
    Bein (gross)
    Tischflaeche

```

STORE();

Input:

```

CompositePart thing("Sitzgarnitur");

```

```

CompositePart sessel("Sessel");
CompositePart tisch("Tisch");
Part bein_klein("Bein (klein)");
Part bein_big("Bein (gross)");
Part sitzfleache("Sitzflaeche");
Part tischfleache("Tischflaeche");

sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&bein_klein);
sessel.addPart(&sitzfleache);

tisch.addPart(&bein_big);
tisch.addPart(&bein_big);
tisch.addPart(&bein_big);
tisch.addPart(&bein_big);
tisch.addPart(&tischfleache);

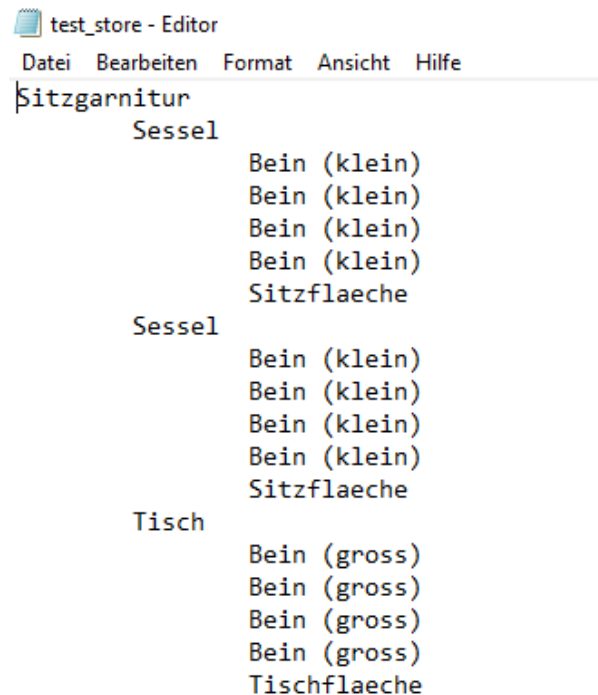
thing.addPart(&sessel);
thing.addPart(&sessel);
thing.addPart(&tisch);

std::ofstream x("test_store.txt");
thing.store(x);

x.close();

```

Output:



```

test_store - Editor
Datei Bearbeiten Format Ansicht Hilfe
Sitzgarnitur
  Sessel
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Sitzflaeche
  Sessel
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Bein (klein)
    Sitzflaeche
  Tisch
    Bein (gross)
    Bein (gross)
    Bein (gross)
    Bein (gross)
    Tischflaeche

```

ZEITAUFWAND

Ca. 20 h