

SWE3-Übung 5

Zeitaufwand in h: 10

Inhaltsverzeichnis

Beispiel 1	2
Lösungsidee	2
Code.....	3
Testfälle	3
Testfall 1	3
Testfall 2	3
Testfall 3	4
Testfall 4	4
Testfall 5	5
Beispiel 2	6
Lösungsidee	6
Code.....	7
Testfälle	7
Testfall 1	7
Testfall 2	8
Testfall 3	9
Testfall 4	9
Testfall 5	10
Testfall 6	11
Testfall 7	11

Beispiel 1

Lösungsidee

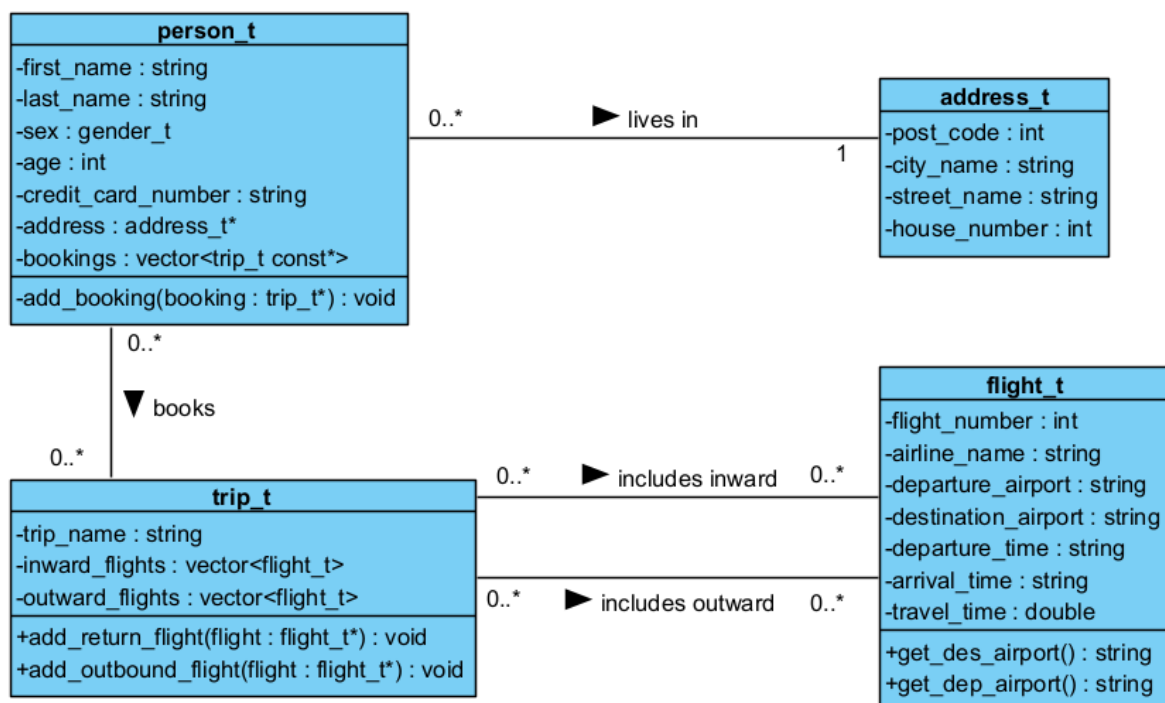
Zu jeder Person (**person_t**) soll eine Adresse mitgespeichert werden. Adressen sollen als eigene Entität (**address_t**) behandelt werden. In einer Adresse können mehrere Personen gemeldet sein.

Personen können mehrere Flugreisen buchen. Diese sollen der jeweiligen Person über eine Methode zugeordnet werden. Eine Flugreise soll dabei auch von mehreren Personen gebucht werden können.

Flugreisen (**trip_t**) haben einen optionalen Namen. Es soll jeweils über eine Methode möglich sein, Flüge zum Hin- oder Rückflug einer Flugreise hinzuzufügen.

Die Flug-Klasse (**flight_t**) soll zwei Getter-Methoden bereitstellen, der Start- bzw. Zielflughafen liefert. Diese werden bei der Flugreise benötigt, um zu zeigen über welche Stationen die Reise geht.

Der Aufbau der vier Klassen wird im folgenden UML-Diagramm gezeigt.



Hin- und Rückflug einer Flugreise beinhalten anfangs keinen Flug, es müssen auch keine angegeben werden. Auch „One-Way-Tickets“ sollen möglich sein.

Wird einer Flugreise kein Name zugeteilt, soll es sich standardmäßig um eine „namenlose Flugreise“ handeln.

Abflug- und Ankunftszeitpunkt sollen als Strings gespeichert werden.

Die Flugdauer wird in Stunden angegeben.

Flüge können selbstverständlich Teil mehrerer unterschiedlicher Flugreisen sein.

Für alle Klassen soll eine einfache Ausgabe möglich sein:

- `flug_t`: gibt alle Details zum entsprechenden Flug aus
- `trip_t`: gibt den Namen aus und welche Flüge für Hin- und Rückflug inkludiert sind
 - o Bsp. für einen Hin- oder Rückweg: Linz -> Mailand -> Kairo.

- address_t: gibt die Adress-Infos aus
- person_t: gibt alle Personendaten aus, inklusive Adresse und allen gebuchten Flugreisen

Um das zu erreichen, muss der Ausgabeoperator überschrieben werden. In den Klassen ist das nicht möglich, weil auf der rechten Seite ein Stream steht. Der Operator wird somit außerhalb der Klasse definiert und implementiert. Um von dort auf die internen Komponenten der Klasse zugreifen zu können, wird das „friend“-Schlüsselwort verwendet. Dabei wird innerhalb der Klasse der entsprechende Funktionsprototyp als „friend“ deklariert, was den Zugriff auf interne Datenkomponenten ermöglicht.

Auf die folgenden Features soll bewusst verzichtet werden:

- Überprüfung, ob der Startflughafen des zum Hin- oder Rückflug einer Flugreise neu hinzugefügten Fluges gleich ist, wie der Zielflughafen des vorherigen Flugs
- Überprüfung, ob die Abflugzeit des neu hinzugefügten Flugs nach der Ankunftszeit des vorherigen Flugs ist.
- Überprüfen, dass zwei Flüge nicht die gleiche Flugnummer haben können.
- Überprüfen, ob ein gültiges Datum bei Abflug- und Ankunftszeit hinterlegt wird.
- Überprüfen, dass die Flugdauer mit Abflug- und Ankunftszeit zusammenpasst (und nicht negativ ist).
- Überprüfen, ob allgemein leere oder ungültige Parameter bei Attributen hinterlegt werden (alle Klassen)
- Möglichkeit des Kopierens von Objekten der einzelnen Klassen

Code

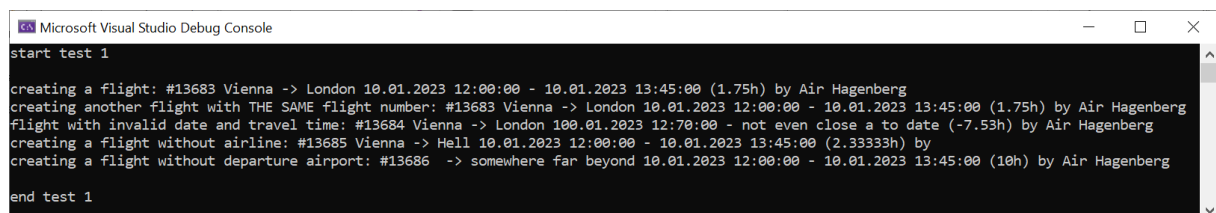
Der Code zu diesem Beispiel befindet sich in „Task1“

Testfälle

Testfall 1

Als erstes werden ein paar Flüge erstellt und ausgegeben. Bei diesem Test sind auch einige der oben genannten Problemfälle getestet:

- Es kann eine Flugnummer mehr als einmal vorkommen
- Abflug- bzw. Ankunftsdatum muss kein gültiges Datum sein
- Ungültige Flugdauer
- Keine Airline oder Flughafen angegeben



```

Microsoft Visual Studio Debug Console
start test 1
creating a flight: #13683 Vienna -> London 10.01.2023 12:00:00 - 10.01.2023 13:45:00 (1.75h) by Air Hagenberg
creating another flight with THE SAME flight number: #13683 Vienna -> London 10.01.2023 12:00:00 - 10.01.2023 13:45:00 (1.75h) by Air Hagenberg
flight with invalid date and travel time: #13684 Vienna -> London 100.01.2023 12:70:00 - not even close a to date (-7.53h) by Air Hagenberg
creating a flight without airline: #13685 Vienna -> Hell 10.01.2023 12:00:00 - 10.01.2023 13:45:00 (2.33333h) by
creating a flight without departure airport: #13686 -> somewhere far beyond 10.01.2023 12:00:00 - 10.01.2023 13:45:00 (10h) by Air Hagenberg
end test 1
  
```

Testfall 2

Hin- und/oder Rückflug einer Flugreise können leer bleiben. Werden Flüge hinzugefügt, kann es sein, dass diese nicht passen, entweder weil der Startflughafen nicht dem Zielflughafen des vorherigen Flugs entspricht, oder weil die Abflugzeit VOR der Ankunftszeit des vorherigen Fluges liegt. Dies wird wie in der Lösungsidee beschrieben, nicht beachtet und so ist folgendes Szenario problemlos möglich, auch dass ein Flug mehrmals eingefügt wird.

```
Microsoft Visual Studio Debug Console

start test 2
Trip without flights: stay at home:
outbound flights: no flights
returning flights: no flights

Only outbound flights: one way ticket!:
outbound flights: Vienna -> London -> Reykjavik
returning flights: no flights

Adding the following flights to return flights:
#98 Reykjavik -> London 20.01.2023 15:00:00 - 20.01.2023 19:45:00 (3.75h) by London Air
#15 Oslo -> Vienna 15.01.2023 12:00:00 - 15.01.2023 13:45:00 (2h) by Air Hagenberg
trip looks as follows: one way ticket!:
outbound flights: Vienna -> London -> Reykjavik
returning flights: Reykjavik -> London -> Vienna

Added the same flights again, check result: one way ticket!:
outbound flights: Vienna -> London -> Reykjavik
returning flights: Reykjavik -> London -> Vienna -> London -> Vienna

end test 2
```

Testfall 3

Beispiele für Adressen:

Auch hier sind ungültige Eingaben möglich wie:

- Negative PLZ oder Hausnummer
- Kein Straßen- oder Stadtname
- Gleiche PLZ aber unterschiedlicher Stadtname

```
Microsoft Visual Studio Debug Console

start test 3

Faschinastrasse 555, 6721 Thueringerberg
Innerberg 666, -6721
Walgaustrasse -777, 6712 Thueringen
Faschinastrasse 888, 6722 St. Gerold
999, 6733 Fontanella

end test 3
```

Testfall 4

Sonderfälle für Personen (noch ohne Buchungen von Flugreisen):

- Kreditkartennummer wird als String abgespeichert, da darf somit irgendwas stehen
- Kein Vor- und/oder Nachname
- Negatives Alter
- Keine Adresse
- Keine Anrede, wenn kein Geschlecht angegeben, sonst: Mr. oder Mrs.

```
Microsoft Visual Studio Debug Console

start test 4

Card No: 004-152-614-141-163
Mr. Friedrich Walter(76) from Walgaustrasse 777, 6712 Thueringen has booked:

Card No: ???
Mrs. Gruber(12) from Faschinastrasse 0, 6733 Fontanella has booked:

Card No: 666-666-666-666-666
Max (-10) from no address given has booked:

Card No: no credit card no :(
Mrs. Reinelde Burtscher(97) from Buchboden 666, 6731 Sonntag has booked:

Card No: 3
Mr. Engelbert Sparr(80) from Faschinastrasse 888, 6722 St. Gerold has booked:

end test 4
```

Testfall 5

Personen mit Flugreisen:

- Alle Flugreisen, die eine Person gebucht hat, werden mit der Person mitausgegeben
- Flugreisen können angepasst/erweitert werden
- Flugreisen können von mehreren Personen gebucht werden
- Verhalten bei „Dangling-Pointern“ soll gezeigt werden

```
Microsoft Visual Studio Debug Console

start test 5

person without bookings: Card No: 234-014-140-414-144
Mr. Ferdinand Maier(69) from Faschinastrasse 555, 6733 Fontanella has booked:

person with one booking without flights
Card No: 234-014-140-414-144
Mr. Ferdinand Maier(69) from Faschinastrasse 555, 6733 Fontanella has booked:
unnamed trip:
outbound flights: no flights
returning flights: no flights

flights have been added to the trip
Card No: 234-014-140-414-144
Mr. Ferdinand Maier(69) from Faschinastrasse 555, 6733 Fontanella has booked:
unnamed trip:
outbound flights: Vienna -> London -> Reykjavik
returning flights: Reykjavik -> London -> Vienna

his wife is joining the trip to Iceland (in January...)
Card No: 234-014-140-414-144
Mrs. Rosmarie Maier(72) from Faschinastrasse 555, 6733 Fontanella has booked:
unnamed trip:
outbound flights: Vienna -> London -> Reykjavik
returning flights: Reykjavik -> London -> Vienna
```

```
Added a new trip:
Card No: 234-014-140-414-144
Mr. Ferdinand Maier(69) from Faschinastrasse 555, 6733 Fontanella has booked:
unnamed trip:
outbound flights: Vienna -> London -> Reykjavik
returning flights: Reykjavik -> London -> Vienna

"Inlandsfluege":
outbound flights: Muenchen -> Palma de Mallorca
returning flights: Palma de Mallorca -> Frankfurt -> Muenchen

Print again:
Card No: 234-014-140-414-144
Mr. Ferdinand Maier(69) from Faschinastrasse 555, 6733 Fontanella has booked:
unnamed trip:
outbound flights: Vienna -> London -> Reykjavik
returning flights: Reykjavik -> London -> Vienna

"Inlandsfluege":
outbound flights:
```

Der Hinflug nach Mallorca wurde in diesem Beispiel am Schluss gelöscht. Das ist schlecht, da so von irgendeiner Speicherstelle etwas gelesen wird, von der wir nichts wissen. Bei diesem Test hatte das Probleme mit der Ausgabe auf die Konsole zur Folge. Weitere Konsolenausgaben wurden danach nicht mehr geschrieben. Egal ob Flug, Flugreise oder Adresse, alle werden bei den hat-Beziehungen über Pointer anderen Klassen übergeben. Bei diesen Beziehungen handelt es sich in keinem Fall um Besitztum, darum muss dort, wo die Instanzen erstellt werden, dafür gesorgt werden, dass die Daten immer gültig sind und länger leben als die Objekte, die diese „haben“.

Beispiel 2

Lösungsidee

Der Name eines „part_t“ oder „composite_part_t“ soll immer mit einem Buchstaben (groß und klein, aber ohne Umlaute), oder mit einer Zahl beginnen und darf danach beliebig viele Buchstaben, Zahlen und gewisse Sonderzeichen enthalten (runde Klammern, Leerzeichen und Bindestrich). Das soll mithilfe eines Regex realisiert werden.

Für ungültige Eingaben (Konstruktor oder Einlesen per File) soll ein Default-Wert für den Namen geschrieben werden. Dieser und das Regex-Pattern für die Überprüfung des Namens soll mithilfe einer statischen Methode innerhalb der Klasse verfügbar sein.

Die Kopier-Funktion für „part_t“ und „composite_part_t“ soll nicht gegeben sein.

Die Getter-Methode „get_name“ liefert immer den aktuellen Namen des „part“ zurück.

Die Equals-Methode prüft zwei Parts auf Gleichheit. Dabei wird es einen Unterschied machen, ob links ein „part_t“ oder ein „composite_part_t“ steht. Steht links ein „part_t“, kann nicht überprüft werden, ob es sich bei der zweiten Instanz um denselben Typ handelt. Es soll in diesem Fall nur der Name verglichen werden. Falls das zweite Objekt ein „composite_part_t“ ist, das möglicherweise noch Parts hat, wird dies ignoriert. Falls das linke Objekt ein „composite_part_t“ ist, kann überprüft werden, ob das rechte Objekt auch (mindestens) ein „composite_part_t“ ist. Ist das nicht der Fall, wird automatisch „falsch“ zurückgeliefert. Sind beides „composite_part_t“ wird zusätzlich zum Namen auch überprüft, ob die Teile, aus denen es besteht, auch alle gleich sind (und deren Unterteile, usw.). Es soll dabei auch darauf geachtet werden, was passiert, wenn „nullptr“ in der Liste sind.

Über eine „add_part“-Methode soll es möglich sein, Teile einem „composite_part_t“ hinzuzufügen.

„get_parts“ ist eine Getter-Methode, um die Liste aller Einzelteile eines „composite_part_t“ zu erhalten. Es soll dabei zwei Varianten von dieser Methode geben. Einmal soll eine konstante Liste, einmal soll eine nicht konstante Liste zurückgegeben werden. Bei der nicht konstanten Variante ist es möglich, die Liste von außen zu bearbeiten. Die konstante Variante wird verwendet, wenn diese nicht abgeändert werden soll, beispielsweise für die Formatter.

Die Implementierung der Formatter soll anders, als in der Aufgabenstellung beschrieben, aussehen.

Laut Aufgabenstellung soll eine abstrakte Klasse „Formatter“ erstellt werden mit der abstrakten Methode „printParts“, die die beiden Klassen „SetFormatter“ und „HierarchyFormatter“ erben. Beide erhalten jedoch ein „part_t“ Element übergeben, das vom jeweiligen Formatter entsprechend ausgegeben wird. Es wird auf keine interne Datenkomponenten zugegriffen. Somit kann die Methode „static“ deklariert werden, was allerdings das Überschreiben bzw. Kennzeichnen als „abstrakte“ Methode unmöglich und die gegebene Klassenhierarchie unnötig macht.

Es wird also stattdessen nur eine statische Klasse „formatter“ erstellt, in dieser sich zwei ebenso statische Funktionen „print_hierarchy“ und „print_part_set“ befinden. Diese sollen das gegebene Objekt auf einen ebenfalls gegebenen Output-Stream schreiben. Die Verwendung einer Klasse macht hierbei Sinn, da beide Funktionen eine rekursive Hilfsfunktion benötigen, die nach außen nicht sichtbar sein soll. Die „print_hierarchy“-Funktion soll alle benötigten Teile mit entsprechenden Einrückungen ausgeben. „print_part_set“ soll zuerst den Namen der „Wurzel“ ausgeben und anschließend alle gleichen „part_t“ zählen. Falls einem „composite_part_t“ keine weiteren Teile zugeordnet sind, wird dies NICHT beim Set hinzugezählt (wirklich NUR alle „part_t“).

„part_t“ soll um ein Interface „storable_t“ erweitert werden, die das Interface von „part_t“ um die Methoden „load“ und „store“ erweitert. Diese soll als abstrakte Klasse implementiert werden, d.h. es kann keine Instanz dieser Klasse erstellt werden.

Die „store“-Methode kann bereits in der abstrakten Klasse ausimplementiert werden. Es wird dabei in erster Linie eine rekursive Funktion aufgerufen, bei der entweder nur der Name des „part_t“ oder der Name des „composite_part_t“ geschrieben wird gemeinsam mit möglichen „Kindern“. Die rekursive „store“ Methode wird ebenso wie die „load“ Methode als Pure-Virtual Methode gekennzeichnet. Ab diesem Moment handelt es sich bei „storable_t“ um eine abstrakte Klasse, von der keine Instanz angelegt werden kann. Die Implementierung der Methoden erfolgt zumindest in der ersten Unterklasse („part_t“). Auch „load“ soll über eine rekursive Hilfsfunktion verfügen. Beide rekursiven Versionen von „load“ und „store“ müssen „public“ gekennzeichnet werden, weil sonst bei einem Pointer-Zugriff auf das Objekt diese Funktionen nicht aufgerufen werden können.

Wird ein „part_t“ in ein File gespeichert, muss nur der Name geschrieben werden. Beim Auslesen soll die erste Nicht-Leerzeile als Name hinterlegt werden. Bereits erwähnte Regeln, damit ein Name gültig ist und Maßnahmen bei nicht Gültigkeit, sollen auch hier angewandt werden.

Beim Speichern eines „composite_part_t“ wird erst der Name geschrieben und anschließend alle Teile der „part_list“ in einem Block mit geschwungenen Klammern. Mögliche Leerzeilen sollen ignoriert werden. Es soll erkannt werden, ob es sich um ein „part_t“, oder ein „composite_part_t“ handelt und im zweiten Falle ein rekursiver Selbstaufruf geschehen. Ein „composite_part_t“ liegt vor, wenn das nächste Element bzw. Zeile eine öffnende, geschwungene Klammer ist. In jedem anderen Fall handelt es sich beim aktuellen Element um ein „part_t“.

Wird ein „composite_part_t“ mit der „load“-Methode geladen, müssen alle Teile (außer die „Wurzel“) mit dem „new“-Schlüsselwort auf dem Heap erstellt werden. Es wird darum gekümmert, dass in diesem Fall, wenn das „Wurzel“ Element stirbt, alle erstellten Teile mitgelöscht werden, um so Speicherleichen zu vermeiden. Aus diesem Grund ist bei manuellen Änderungen an dem Objekt oder bei Weiterverarbeitung einzelner Elemente äußerste Vorsicht zu nehmen! Wird das „composite_part_t“ nicht über die „load“-Methode initialisiert, werden die einzelnen Teile nicht gelöscht, da ihm diese nicht gehören. Das ist eigentlich auch der Normalfall.

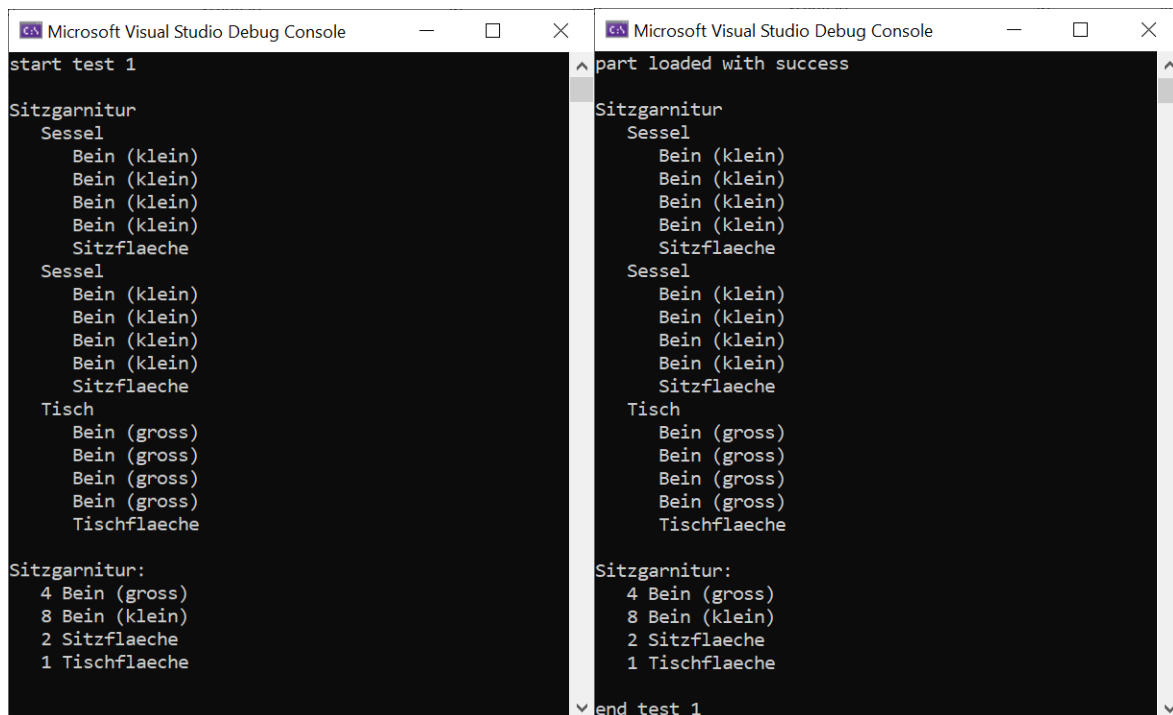
Code

Der Code zu diesem Beispiel

Testfälle

Testfall 1

Beispiel aus Angabe. Wird anschließend in ein File geschrieben und wieder ausgelesen und auf Gleichheit überprüft (ob das ausgelesene Zusammenbauteil gleich dem gespeicherten ist):



```
start test 1
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

Sitzgarnitur:
  4 Bein (gross)
  8 Bein (klein)
  2 Sitzflaeche
  1 Tischflaeche

part loaded with success
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

Sitzgarnitur:
  4 Bein (gross)
  8 Bein (klein)
  2 Sitzflaeche
  1 Tischflaeche

end test 1
```

Testfall 2

Bei diesem Testfall werden zuerst ungültige Teile-Namen eliminiert.

Danach wird ein Zusammenbau-Teil erstellt und öfter abgespeichert und wieder ausgelesen und verglichen in den folgenden Kombinationen (gespeichert als => ausgelesen als):

- composite part => composite part
- composite part => part
- part => part
- part => composite part

Das „part“ wird immer als einzelner String abgespeichert und als normales „part“, bei dem nur die erste Zeile berücksichtigt wird, eingelesen. Jedes „composite part“ wird entweder normal eingelesen, wenn nur ein String im File steht, dann bleibt die Liste der Einzelteile einfach leer. Besondere Fälle gibt es beim Vergleich, wenn die beiden gegenüberstehenden Teile von einen unterschiedlichen (dynamischen) Datentyp sind. Dann hängt es davon ab, welches davon links steht. Steht links das „composite part“, dann erkennt dieses, dass rechts ein normales „part“ ist. In diesem Fall sind die beiden Teile immer ungleich! Steht links das „part“ erkennt dieses nicht, was auf der rechten Seite steht und es wird somit nur der Name verglichen und dementsprechend „wahr“ oder „falsch“ zurückgegeben.


```
Microsoft Visual Studio Debug Console

start test 2

creating composite part: invalid part name detected! taking default name
new part name: undefined part
creating composite part: invalid part name detected! taking default name
new part name: undefined part
creating composite part: invalid part name detected! taking default name
new part name: undefined part
creating composite part: invalid part name detected! taking default name
new part name: undefined part
creating composite part: invalid part name detected! taking default name
new part name: undefined part
creating composite part: invalid part name detected! taking default name
new part name: undefined part
creating composite part with valid name: new part name: 1()

savig part...
loading part...
check if equal... they are equal!

loading stored composite part as part...
check if equal... They are not equal because (empty) composite part != part
saving part...
loading part...
check if equal... they are equal!

creating composite part... part name: hello
now load previously stored part as composite part...
check if equal... THEY ARE EQUAL (part == composite part!!!)

part_t: 1()
composite_part_t: 1()

end test 2
```

Testfall 3

Testfall zeigt die Formatter-Ausgabe auch bei „composite part“ ohne gegebener Einzelteile und nur für einen „part“.

```
Microsoft Visual Studio Debug Console

start test 3

hierarchy:
Computermaus

sets:
Computermaus: no composite parts

updating part:

hierarchy:
Computermaus
  Computermuskabel
  Computermausmaus

sets:
Computermaus:
  1 Computermuskabel
  1 Computermausmaus

output for part_t:

hierarchy:
Computermuskabel

sets:
Computermuskabel: no composite parts

end test 3
```

Testfall 4

Als nächstes soll getestet werden, was passiert, wenn die Liste mit allen Teilen ausgelesen wird und dieser anschließend ein „nullptr“ hinzugefügt wird:

```
Microsoft Visual Studio Debug Console

sets:
Kletterausruestung:
  1 60m Halbseil
  1 Bandschlinge
  1 Chalk
  1 Ersatz-Chalk
  1 Erste-Hilfe-Set
  12 Express-Schlinge
  4 Karabiner
  1 Klettergurt natuerlich
  1 Notfall-Chalk
  1 Reepschnur
  1 Steinschlaghelm
  1 linker Kletterpatschen
  1 mobile Sicherungen
  1 rechter Kletterpatschen
  1 und viel mehr

saved and loaded, now check if they are equal:

They are not equal yet...

adding a nullptr like for the original part...

They are equal now!

end test 4
```

Alle Operationen funktionierten weiterhin, allerdings geht der „nullptr“ verloren, wenn ins File geschrieben wird und die „Equals“-Funktion vergleicht, ob dieser „nullptr“ bei beiden zu vergleichenden Teilen an derselben Position steht.

Testfall 5

Nächster Sonder- bzw. Problemfall: Wenn ein „composite part“ aus einem File gelesen wird, müssen alle Teile, die dafür benötigt werden, mit dem „new“-Schlüsselwort auf dem Heap erstellt werden. Ist das der Fall müssen alle Teile beim Zerstören des Elements gelöscht werden (Destruktor). Wenn jetzt neue Bauteile hinzugefügt werden, werden diese auch zerstört. Werden diese nicht auf dem Heap erstellt, stürzt das Programm ab. Werden sie am Heap erstellt, darf nicht vergessen werden, dass die Variable hinter den Pointern stirbt, sobald das „composite part“ dazu stirbt.

```
Microsoft Visual Studio Debug Console

kleine Pfanne

sets:
Bratpfannenset:
  1 kleine Pfanne
  1 mittlere Pfanne
  1 riesige Pfanne

hierarchy:
tolle Pfanne

sets:
tolle Pfanne: no composite parts

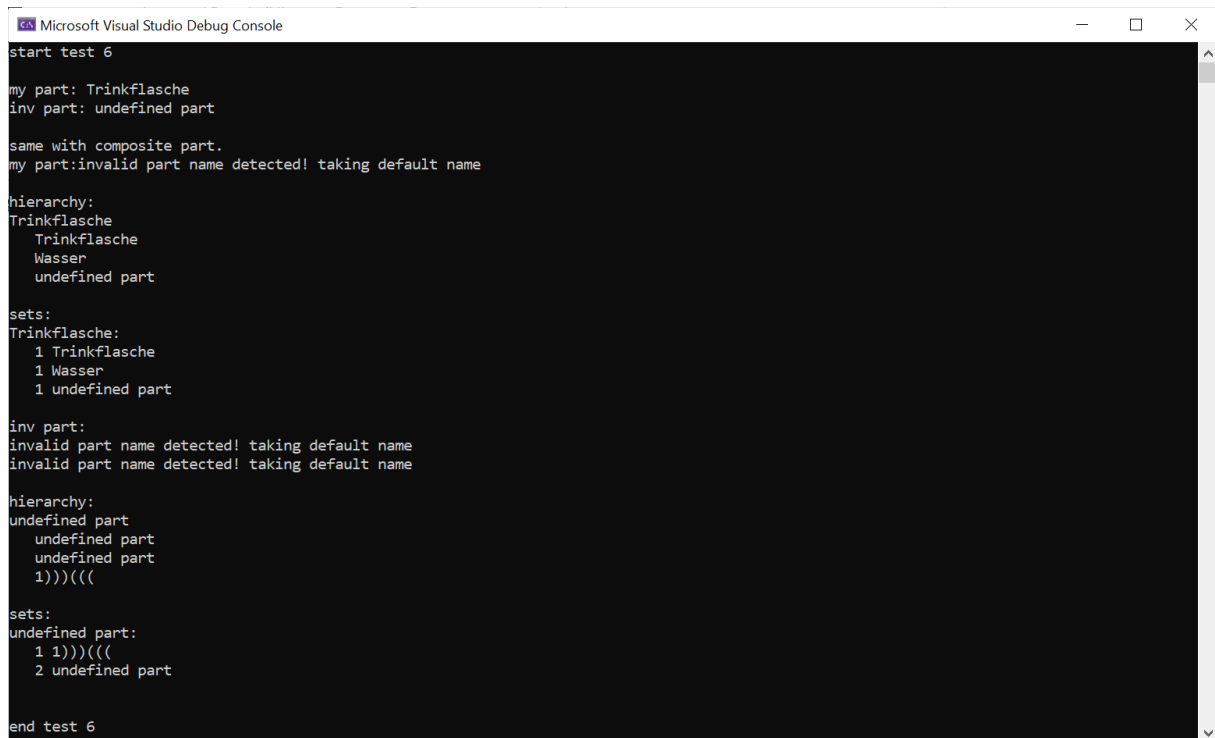
hierarchy:
Bratpfannenset
  riesige Pfanne
  mittlere Pfanne
  kleine Pfanne
  tolle Pfanne

sets:
Bratpfannenset:
  1 kleine Pfanne
  1 mittlere Pfanne
  1 riesige Pfanne
  1 tolle Pfanne

end test 5
```

Testfall 6

In diesem Testfall werden fehlerhaften Eingaben in Files getestet. In jedem Fall werden Leerzeichen ignoriert und ungültige Eingaben erkannt und zu „undefined part“ geändert.

The screenshot shows the Microsoft Visual Studio Debug Console with a black background and white text. The output of 'start test 6' is as follows:

```
start test 6

my part: Trinkflasche
inv part: undefined part

same with composite part.
my part:invalid part name detected! taking default name

hierarchy:
Trinkflasche
  Trinkflasche
  Wasser
  undefined part

sets:
Trinkflasche:
  1 Trinkflasche
  1 Wasser
  1 undefined part

inv part:
invalid part name detected! taking default name
invalid part name detected! taking default name

hierarchy:
undefined part
  undefined part
  undefined part
  1)))(((

sets:
undefined part:
  1 1)))(((
  2 undefined part

end test 6
```

Testfall 7

Dieser Testfall soll zeigen, dass nur jene Teile bei der Set-Übersicht ausgegeben werden, die auch wirklich „part_t“ sind. „composite_part_t“ ohne „Sub-Parts“ werden nicht ausgegeben:

The screenshot shows the Microsoft Visual Studio Debug Console with a black background and white text. The output of 'start test 7' is as follows:

```
start test 7

hierarchy:
epas
  hey
  ho
  haaa

sets:
epas:
  1 ho

end test 7
```

In diesem Fall wurden die Daten aus einem File ausgelesen. „hey“ und „haaa“ sind dort als „composite_part_t“ hinterlegt und werden bei den Sets nicht berücksichtigt.