Übung 05

8h 50min

Beispiel 1

Lösungsidee

Die Klasse Person enthält die in der Angabe gelisteten Attribute. Da diese alle privat sind gibt es getter dafür. Die Klasse Flug zusätzlich zu den in der Angabe gelisteten Attribute einen Vektor von Passagieren (Personen) und hat auch für alle Attribute getter bzw setter. Eine Flugreise besteht aus einer Liste für die Hinflüge, einer für die Rückflüge und dem Start (und somit auch dem Ziel) der Reise. Für jede der beiden Listen gibt es eine add Methode. add_hinflug bzw. add_rückflug erhalten ein Objekt der Klasse Flug per reference und fügt dieses hinten der Liste an, wenn es noch kein Objekt mit derselben Flugnummer gibt.
operator<< gibt für den Hinflug als ersten Ort den Start aus, danach die Elemente der Hinflugliste, bei der Rückflugliste werden zuerst die Elmente dieser ausgegeben und danach der Startort, da man am Ende ja dort wieder ankommen sollte.

Quelltext

Siehe SWE3_Waser_UE05_Bsp1

Testfälle

```
Test Person:
with default constructor:
get_vorname:
get_nachname:
get_alter:      0
get_adresse:
get_geschlecht: weiblich
get_kreditkarten_nummer:

constructor with values:
get_vorname:    Testine
get_nachname:   Testinger
get_alter:      21
get_adresse:    Adresse 23, 1234 Stadt
get_geschlecht: weiblich
get_kreditkarten_nummer:        12345666677899

use 'setter' and change vorname and alter:
get_vorname:    Tina
get_alter:      22

Test Flug:
 default consturctor:
get_flugnummer:
get_fluggesellschaft:
get_abflug:
get_ankunft:
get_dauer:      0
get_ort:

Flug with values:
get_flugnummer: 1234A
get_fluggesellschaft:   Fluggesellschaft
get_abflug:     12:00
get_ankunft:    14:00
get_dauer:      120
get_ort:        Wien
get_passagiere:
Testine Testinger
Testian Testinger
Test Testner

test Flugreise:
Hinflug: Linz -> Frankfurt -> Denver -> Las Vegas
Rueckflug: Las Vegas -> San Francisco -> M³nchen -> Linz
try add same flug twice: Hinflug: Linz -> Frankfurt -> Denver -> Las Vegas -> San Francisco
Rueckflug: Las Vegas -> San Francisco -> M³nchen -> Denver -> Linz
```

Beispiel 2

Lösungsidee

IStorable

Interface, welches die Methoden store und load vorgibt. load ist privat, da es später immer nur in den Konstruktoren aufgerufen werden soll

Part

Die Klasse Part besitzt als Attribut nur *name*, auf welchen über die Methode getName zugegriffen werden kann.

Konstruktoren es gibt 3 verschiedene: einen Default-Konstruktor in welchem der Name auf „unknown part“ gesetzt wird, einen der einen übergebenen String als Namen setzt und einen der einen Inputstream übergeben bekommt und damit load aufruft.

equals erhält ein anderes Part-Objekt und vergleicht die Namen dieser und returniert das Ergebnis des Vergleichs

load liest eine Zeile des Inputstreams und setzt diese als Namen

store schreibt den Namen auf einen übergebenen Outputstream

CompositePart

Erbt von Part und erweitert diese Klasse um einen Vektor von Part-Pointern in welchem die Einzelteile eines CompositeParts gespeichert werden.

Konstruktoren besitzt die gleichen 3 Arten von Konstruktoren wie Part (nur leicht abgeändert)

getParts liefert den Parts-Vector

addPart fügt einen Part-Pointer dem Parts-Vector hinzu

store schreibt den Namen des CompositeParts auf einen Outputstream und ruft, falls der Part-Vector mindestens ein Element besitzt, für diese ebenfalls die store-Method auf um diese auch auf den Stream zu schreiben. Vor der Ausgabe der Einzelteile wird eine öffnende geschwungene Klammer ausgegeben und danach, in einer Leerzeile, eine schließende

load ruf load_rec auf und setzt danach den Namen und kopiert die Vektorelemente

load_rec liest nach der Überprüfung des Inputstreams eine Zeile ein und sucht in dieser mit Hilfe der Funktion *find* nach einer öffnenden bzw. schließenden Klammer. Wird eine schließende gefunden, wird ein nullptr returniert, da sich in dieser Zeile dann kein Part befinden kann. Wird eine öffnende Klammer gefunden wird alles bis zu dieser als Name für einen CompositPart verwendet und solange load_rec aufgerufen und die Ergbnisse dem Parts-Vector dieses neuen CompositeParts hinzugefügt, bis ein nullptr erhalten wird, dann wird ein Pointer auf diesen CompositePart zurückgegeben. Enthält eine Zeile keine Klammer wird ein Pointer auf ein Part-Objekt mit der Zeile als Namen zurückgegeben.

Beispiel für die Struktur eines Files zum Einlesen:
CompositePart name{
anderer CompositePart name{
Part
Part
}
Part
}

Formatter
Ist eine abstrakt Klasse die die Methode printParts vorgibt. Damit die Klasse abstrakt ist, muss printParts eine pure virtual function sein (also hinten = 0). Da printParts dynamisch gebunden wird, sollte es auch einen dynamisch gebundenen Destruktor geben

HierarchyFormatter
Erbt von Formatter und überschreibt die printParts-Methode. printParts ruft jedoch printPartsRec auf.

printPartsRec bekommt einen Pointer auf einen Part und das level (Anzahl der Einrückungen) übergeben. Zuerst werden die Einrückungen als Tabulatoren auf die Konsole geschrieben, danach wird über getName der Name des Parts ausgegeben. Mit einem dynamic-cast wird überprüft, ob es sich bei dem Part-Pointer um einen Pointer auf einen CompositePart handelt. Ist das der Fall, wird für jedes Element des Part-Vektors printPartsRec mit einer Einrückung mehr aufgerufen.

SetFormatter
Erbt genau wie HierarchyFormatter von Formatter und überschreibt printParts.

printParts Zuerst ein dynamic cast durchgeführt und gegebenenfalls für die Elemente des Part-Vektors countParts aufgerufen. Danach wird der Name des Objekts auf welches der Pointer zeigt ausgegeben. Anschließend werden die Elemente der Map ausgegeben. Key ist hierbei der Name eines Teils und Value die Anzahl an vorkommen im Part-Vektor bzw. den Part-Vektoren der CompositeParts.

countParts Erhält einen Pointer auf ein Part-Objekt und überprüft mit einem dynamic cast ob es sich um einen CompositePart handelt. Ist dem der Fall, wird für alle Elemente des Part-Vektors wiederum countParts aufgerufen. Handelt es sich um einen Part wird überprüft, ob dieser bereits in der partCounts Map befindet oder nicht und es wird die Anzahl der Vorkommen erhöht bzw. ein neues Element mit Wert 1 hinzugefügt.

Quelltext
Siehe SWE3_Waser_UE05_Bsp2

Testfälle
Test der Konstruktoren (bis auf die, die einn Inputstream verwenden, dieser werden in testLoad getestet)

```
Test Constructors       Expected        Actual
Default (Part)          unknown part    unknown part
Default(Comp.Part)      unknown part    unknown part
given string (Part)     Test test       Test test
given string (Comp.Part) Test test      Test test
```

Test equals für Part und CompositePart

```
Test equals:
compare two composite parts: test1 ('Sessel') and test2 ('Sessel rot')
Sessel
        4       Bein (klein)
        1       Sitzflaeche
Sessel rot
        4       Bein (klein)
        1       Sitzflaeche
test1.equals(test2)     expected: false actual: false
test2.equals(test1)     expected: false actual: false

compare two parts: test3 ('Tischflaeche') and test4 ('Tischflaeche')
Tischflaeche
Tischflaeche
test3.equals(test4)     expected: true  actual: true
test4.equals(test3)     expected: true  actual: true

compare composite part and part: test1 ('Sessel') and test4 ('Tischflaeche')
test1.equals(test4)     expected: false actual: false
test4.equals(test1)     expected: false actual: false

compare part to itself: test4 ('Tischflaeche')
test4.equals(test4)     expected: true  actual: true
test4.equals(test4)     expected: true  actual: true

compare composite part to itself: test2 ('Sessel rot')
test2.equals(test2)     expected: true  actual: true
test4.equals(test4)     expected: true  actual: true
compare two default parts       expected: true actual: true

compare two default composite parts     expected: true actual: true
```

Test von addPart und getParts

```
Test addPart and getParts
create composite part 'Sessel', print with SetFormatter:
Sessel

add part ('Bein') four times, print with SetFormatter:
Sessel
        4       Bein
getParts and change name of second part to ('Bein klein'): (print with HierarchyFormatter)
Sessel
        Bein
        Bein klein
        Bein
        Bein

add composite part (print with HierarchyFormatter)
Sessel
        Bein
        Bein klein
        Bein
        Bein
        Kissen
                Kissenbezug
                Fuellkissen
```

Test HierarchyFormatter:

```
test HierarchyFormatter:
Test with CompositPart:
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
        Vase

Test with CompositPart without any elements in Part-Vector:     unknown part

Test with Part: Bein (klein)

Test with default-Part: unknown part
```

Test SetFormatter:

```
test SetFormatter:
Test with CompositPart:
Sitzgarnitur
        4       Bein (gross)
        8       Bein (klein)
        2       Sitzflaeche
        1       Tischflaeche
        1       Vase
Test with CompositPart without any elements in Part-Vector:     unknown part
Test with Part: Bein (klein)
Test with default-Part: unknown part
```

Test store:

```
test store
stored composite in test_store_composite.txt
stored part in test_store_part.txt
store default part in test_store_part_default.txt
stored default composite part in test_store_composite_default.txt
stored composite part without parts in test_store_composite_empty.txt
try storing part with closed stream in test_closed.txt
```

Inhalt der Files:

test_store_part_default.txt - Editor

Datei Bearbeiten Ansicht

```
unknown part
```

test_store_composite_default.txt - Editor

Datei Bearbeiten Ansicht

```
unknown part
```

test_store_composite.txt - Editor

Datei Bearbeiten Ansicht

```
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
}
Vase
}
```

test_store_composite_empty.txt - Editor

Datei Bearbeiten Ansicht

```
Schreibtisch
```

test_closed.txt - Editor

Datei Bearbeiten Ansicht

test_store_part.txt - Editor

Datei Bearbeiten Ansicht

```
Tischflaeche
```

Test load:

```
test load: load Part/Compositepart then print with HierarchyFormatter (HF) and SetFormatter(SF)
load CompositePart from test_store_composite.txt:
HF:
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
        Vase
SF:
Sitzgarnitur
        4       Bein (gross)
        8       Bein (klein)
        2       Sitzflaeche
        1       Tischflaeche
        1       Vase

load CompositePart from test_store_composite_default.txt:
HF:
unknown part
SF:
unknown part

load CompositePart from test_store_composite_empty.txt:
HF:
Schreibtisch
SF:
Schreibtisch

load CompositePart from test_store_part.txt:
HF:
Tischflaeche
SF:
Tischflaeche

load Part from test_store_part.txt:
HF:
Tischflaeche
SF:
Tischflaeche
```

```
load Part from test_store_part_default.txt:
HF:
unknown part
SF:
unknown part

load Part from test_store_composite.txt:
HF:
Sitzgarnitur{
SF:
Sitzgarnitur{

load Part from test_store_composite_empty.txt:
HF:
unknown part
SF:
unknown part

try load Part from closed stream:
HF:

SF:


try load Part from empty file:
HF:

SF:
```