SWE3_Tober_Ue5

Beispiel 1

Lösungsidee:

Es sollen drei Klassen implementiert werden: Person, Flug und Flugreise. Die Klasse Person enthält diverse private Datentypen (Vorname, Nachname, Geschlecht, Alter, Adresse, Kreditkartennummer), mit entsprechenden öffentlichen Methoden soll man darauf zugreifen können. Die Klasse Flug enthält privaten Datentypen Flugnummer, Fluggesellschaft, Ort, Ablfug- und Ankunftszeit, sowie Flugdauer. Diese können wieder mit öffentlichen Methoden aufgerufen werden. Zusätzlich macht es Sinn, für jeden Flug, eine Liste von Passagieren (Person) zusammenzustellen, die mitfliegen. Eine Flugreise enthält dieselben Datentypen wie Flug + eine Liste von Flügen, die entweder zu dem Hinflug oder zum Rückflug gehören. Für dieses Beispiel wird keine Vererbung benötigt. Die Wahl der privaten Datentypen sollte sinnvoll gewählt werden und ruft man beispielsweise nur “Person a;“ ohne Parameter auf, so kann man ausgeben, dass eben noch keine Person, Flug, Flugreise existiert, da diese mit Parametern befüllt werden sollte zur sinnvollen Nutzung. Für das Überladen des <<-Operators kann dieser außerhalb der Klasse definiert werden und mit Hilfe einer vordefinierten Print-Funktion für Flugreisen implementiert werden.

Testfälle:

Testfall 1: Testen der Methoden der Klasse Person

```
First name: Maria
Last name: Mustermann
Gender: f
Age: 21
Adress: 4240 Freistadt, Taubenstra e 69
Creditcard number: 1234

First name: Fabian
Last name: Tober
Gender: m
Age: 22
Adress: 4240 Freistadt, Taubenstra e 6
Creditcard number: 1111

First name: Lukas
Last name: Hartmann
Gender: f
Age: 14
Adress: 4240 Freistadt, Grosslingen 2
Creditcard number: 5555
```

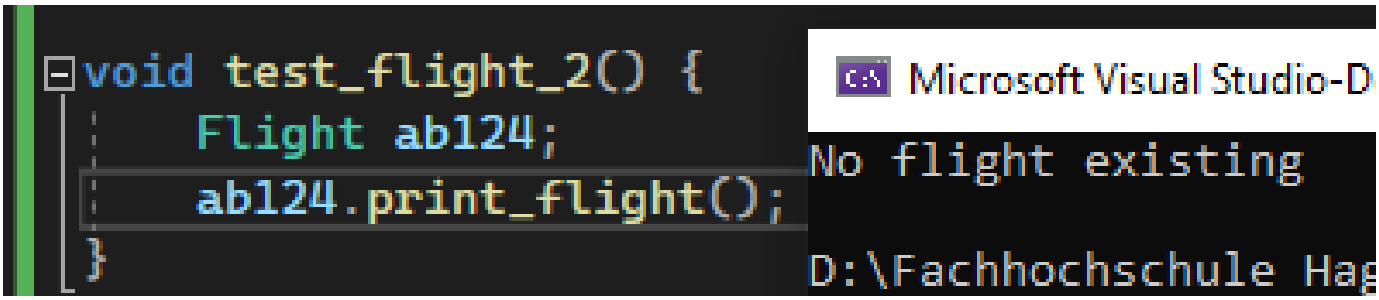
```
void test_person_2() {
    Person d;
    d.print_person();
}
```

```
Microsoft Visual Studio-Debugging-Konsole
No person found for this flight

D:\Fachhochschule Hagenberg\3. Se
el 1\x64\Debug\Beispiel 1.exe (Pr
```

Testfall 2: Testen der Methoden der Klasse Flight

```
Flight number: AB123
Flight company: Hagenberg Airlines
Location: Hagenberg
Departure time: 14:00
Arriving time: 16:00
Flight duration: 120 minutes
Travel list including first name, last_name and overall amount of travellers for this flight:
Maria Mustermann
Fabian Tober
Lukas Hartmann
Overall amount of travellers for this flight: 3
```



Testfall 3: Testen der Methoden der Klasse AirTravel

```
utward flights:
light number: AB123
light company: Hagenberg Airlines
ocation: Hagenberg
eparture time: 14:00
rriving time: 16:00
light duration: 120 minutes
ravel list including first name, last_name and overall amount of travellers for this flight:
aria Mustermann
abian Tober
ukas Hartmann
verall amount of travellers for this flight: 3




eturn flights:
light number: AX451
light company: Hagenberg Airlines
ocation: Toronto
eparture time: 14:00
rriving time: 16:00
light duration: 120 minutes
ravel list including first name, last_name and overall amount of travellers for this flight:
lara Kauffrau
ans Wurst
oritz Strolch
verall amount of travellers for this flight: 3



light number: XZ986
light company: Hagenberg Airlines
ocation: Salzburg
eparture time: 01:00
rriving time: 04:00
light duration: 240 minutes
ravel list including first name, last_name and overall amount of travellers for this flight:
aria Mustermann
ukas Hartmann
oritz Strolch
aria Mustermann
verall amount of travellers for this flight: 4
```

```
void test_air_travel_2() {
    AirTravel air_travel_2;
    std::cout << air_travel_2;
}
```

```
Microsoft Visual Studio-Debugging-Konso
Not outward flights existing
Not return flights existing
Outward flights:



Return flights:

D:\Fachhochschule Hagenberg\3.
```

Beispiel 2

Lösungsidee:

Es soll ein Programm in C++ für eine Stücklistenverwaltung implementiert werden. In einem Namespace PartsLists gibt es eine Klasse Part mit einem privaten Attribut name, einer Zugriffsmethode getName(), einem Konstruktor und einer Vergleichsmethode. Die Klasse CompositePart ist eine Klasse, die von der Klasse Part erbt. Sie enthält ebenfalls einen Konstruktor, eine Methode zum Hinzufügen eines Teils add_part() und eine Funktion ge_parts(), die einen Vektor zurückliefert, mit der man alle Teile, die im zusammengesetzten Teil enthalten sind, bearbeiten kann.

Des Weiteren gibt es noch eine Klasse Formatter, die eine Methode print_parts() enthält. Die beiden Methoden SetFormatter und HierarchyFormatter erben jeweils von der Formatter Klasse und da diese beiden Unterklassen eine gleichnamige Methode print_parts() haben, werden diese überschrieben, also die Ausgangsmethode der Klasse Formatter wird nicht weiter definiert.

Die print-methode des SetFormatters gibt alle Teile eines Zusammengesetzten Teils aus. Die print-methode des HierarchyFormatters gibt pro Teil nur den Namen und die Anzahl des jeweiligen Teils aus.

Zusätzlich gibt es eine Klasse Storable, die zwei Methoden store() und load() bereitstellt, die eben Stücklisten laden und speichern können.

Testfälle:

Testfall 1: Testen der Methoden der Klasse Part

```
void test_constructor() {
    Part a;
    cout <<  "Name of this part: " << a.get_name();
}
```

```
Microsoft Visual Studio-Debugging
Name of this part: no name
D:\Fachhochschule Hagenberg
5\x64\Debug\Beispiel 2.exe
```

```
void test_constructor_2() {
    Part b("sessel");
    cout << "Name of this part: " << b.get_name();
}
```

```
Microsoft Visual Studio-Debugging
Name of this part: sessel
D:\Fachhochschule Hagenberg
5\x64\Debug\Beispiel 2.exe
```

```
Part 1: sessel
Part 2: Sessel
Both parts are not the exact same
```

```
Part 1: sessel
Part 2: sessel
Both parts are the exact same
```

Testfall 2: Testen der Methoden der Klasse CompositePart

```
void test_composite_part_constructor() {

    CompositePart* sitzgarnitur = new CompositePart("Sitzgarnitur");
    cout << "Composite part: " << sitzgarnitur->get_name() << endl;
    delete sitzgarnitur;
}
```

```
Microsoft Visual Studio-Debugging-Kon
Composite part: Sitzgarnitur

D:\Fachhochschule Hagenberg\3.
```

```
void test_add_part(){
    CompositePart* sitzgarnitur = new CompositePart("Sitzgarnitur");
    Part a("sessel");
    Part b("tisch");
    Part c("tür");

    sitzgarnitur->add_part(a);
    sitzgarnitur->add_part(b);
    sitzgarnitur->add_part(c);

    cout << "Size of composite part: " << sitzgarnitur->getParts().size();
}
```

```
Microsoft Visual Studio-Debugging-
Size of composite part: 3
D:\Fachhochschule Hagenberg\
5\x64\Debug\Beispiel 2.exe (
Um die Konsole beim Beenden
"Konsole beim Beenden des De
Drücken Sie eine beliebige T
```

```
void test_get_parts() {
    CompositePart* sitzgarnitur = new CompositePart("Sitzgarnitur");
    Part a("sessel");
    Part b("tisch");
    Part c("tür");

    sitzgarnitur->add_part(a);
    sitzgarnitur->add_part(b);
    sitzgarnitur->add_part(c);

    cout << "Composite Part: " << sitzgarnitur->get_name() << endl;
    cout << "Parts of composite part: " << endl;
    for (size_t i{ 0 }; i < sitzgarnitur->getParts().size(); i++ ){
        cout << &sitzgarnitur->getParts()[i] << endl; //just getting
    }
```

```
Microsoft Visual Studio-Debugging-Ko
Composite Part: Sitzgarnitur
Parts of composite part:
0000024DC170C0B0
0000024DC170BBD8
0000024DC170C120

D:\Fachhochschule Hagenberg\3
5\x64\Debug\Beispiel 2.exe (P
Um die Konsole beim Beenden d
"Konsole beim Beenden des Deb
Drücken Sie eine beliebige Ta
```

Testfall 4: Testen der beiden Formatter

Testfall 5: Testen der Methoden der Klasse Storable

Allgemeine Anmerkungen:

Aufwand in Stunden: 6h