

Beispiel 1: Flugreisen

Lösungsidee:

In dem ersten Beispiel wurden mehrere Klassen implementiert, die zusammen ein System zur Verwaltung von Flugreisen darstellen. Die Klasse `adress_t` repräsentiert eine Adresse und enthält Membervariablen für Straße, Hausnummer, Postleitzahl, Stadt und Land. Es gibt auch Konstruktoren und eine `to_string`-Methode, die die Adresse in Form eines Strings ausgibt.

Die Klasse `Person` repräsentiert eine Person und enthält Membervariablen für den Vornamen, Nachnamen, Geschlecht, Adresse, Alter und Kreditkartennummer. Es gibt auch eine `to_string`-Methode, die die Person in Form eines Strings ausgibt. Das Geschlecht der Person wird als Wert des Enums `gender_t` gespeichert, das vier mögliche Optionen hat: männlich, weiblich, nicht-binär und keine Angabe.

Die Klasse `Flug` repräsentiert einen Flug und enthält Membervariablen für Flugnummer, Abflughafen, Ziel, Abflugzeit, Ankunftszeit, Flugdauer und Fluggesellschaft. Es gibt auch eine `to_string`-Methode, die den Flug in Form eines Strings ausgibt. Die Flugzeiten werden als `time_t`-Werte gespeichert, um eine automatische Berechnung der Flugdauer zu ermöglichen.

Die Klasse `Flugreise` repräsentiert eine Flugreise und enthält Membervariablen für Listen von Passagieren und Flügen (Hinflüge und Rückflüge). Es gibt auch eine `to_string`-Methode, die die Flugreise in Form eines Strings ausgibt, und eine `<<`-Operatorüberladung, die es ermöglicht, eine Flugreise direkt in einen ostream auszugeben. Es gibt auch Methoden zum Hinzufügen von Flügen und Passagieren zu einer Flugreise.

Der Enum `flight_t` wird verwendet, um anzugeben, ob ein Flug ein Hinflug oder ein Rückflug ist.

Zusammengefasst bietet dieser Code eine strukturierte Möglichkeit, Flugreisen und die dazugehörigen Flüge und Passagiere zu verwalten. So kann man Flugreisen erstellen, Flüge und Passagiere hinzufügen, Flugreisen in Form von Strings ausgeben und Informationen über Flüge und Passagiere abrufen.

Testfälle:

Es wird davon ausgegangen dass die bereitgestellten Funktionen gemäß ihrer Funktionsdefinition die richtigen Argumente erhalten. Wird z.B bei einer Strasse eine Zahl statt einem String übergeben, ist mit Fehlern zu rechnen.

```
//Test von Grundfunktionalitäten
void test1() {
    //Erstellen der Adressen
    adress_t adress("Gute Strasse", 1, 4580, "Vienna", "Austria");

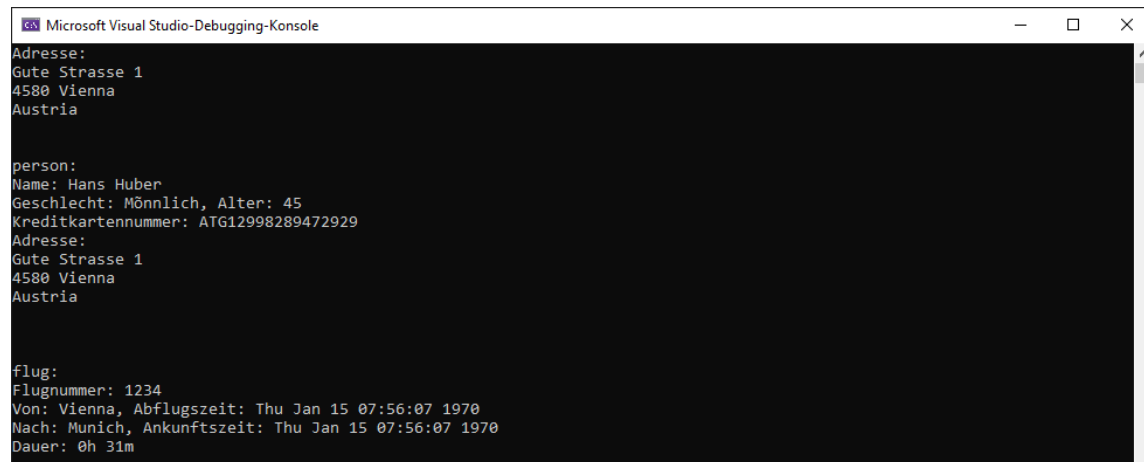
    //Person anlegen
    Person person1("Hans", "Huber", gender_t::male, adress, 45, "ATG12998289472929");

    //Flug anlegen
    Flug flug1(1234, "Vienna", "Munich", 1234567, 1236468, "Lufthansa");

    //Adresse ausgeben
    std::cout << adress.to_string();

    //Person ausgeben
    std::cout << "\n\nperson:\n";
    std::cout << person1.to_string();

    //Flug ausgeben
    std::cout << "\n\nflug:\n";
    std::cout << flug1.to_string();
}
```



```
Microsoft Visual Studio-Debugging-Konsole

Adresse:
Gute Strasse 1
4580 Vienna
Austria

person:
Name: Hans Huber
Geschlecht: Männlich, Alter: 45
Kreditkartennummer: ATG12998289472929
Adresse:
Gute Strasse 1
4580 Vienna
Austria

flug:
Flugnummer: 1234
Von: Vienna, Abflugszeit: Thu Jan 15 07:56:07 1970
Nach: Munich, Ankunftszeit: Thu Jan 15 07:56:07 1970
Dauer: 0h 31m
```

```
//Eine Familienreise mit Pärchen und Sohn
void test2() {
    Flugreise reise1;

    //Erstellen der Adressen
    adress_t adress1("Gute Strasse", 1, 4580, "Vienna", "Austria");
    adress_t adress2("Neue Strasse", 3, 6969, "Hagenberg", "Austria");

    //Personen anlegen
    Person person1("Hans", "Huber", gender_t::male, adress1, 45, "ATG12998289472929");
    Person person2("Frederike", "Huber", gender_t::female, adress1, 42, "ATG143636456456");
    Person person3("Max", "Huber", gender_t::male, adress2, 24, "ATG143636456456");

    //Flüge anlegen
    Flug flug1(1234, "Vienna", "Munich", 1234567, 1236468, "Lufthansa");
    Flug flug2(1246, "Munich", "Amsterdam", 1246468, 1256468, "Lufthansa");

    //Die Personen zur Reise Hinzufügen
    reise1.add_passenger(person1);
    reise1.add_passenger(person2);
    reise1.add_passenger(person3);

    //Die Flüge zur Reise hinzufügen
    reise1.add_flight(flug1, flight_t::outwards);
    reise1.add_flight(flug2, flight_t::outwards);

    //Die gesamte Reise ausgeben
    std::cout << reise1;
}
```

Microsoft Visual Studio-Debugging-Konsole

Name: Hans Huber
Geschlecht: Männlich, Alter: 45
Kreditkartennummer: ATG12998289472929
Adresse:
Gute Strasse 1
4580 Vienna
Austria

Name: Frederike Huber
Geschlecht: Weiblich, Alter: 42
Kreditkartennummer: ATG143636456456
Adresse:
Gute Strasse 1
4580 Vienna
Austria

Name: Max Huber
Geschlecht: Männlich, Alter: 24
Kreditkartennummer: ATG143636456456
Adresse:
Neue Strasse 3
6969 Hagenberg
Austria

Flugnummer: 1234
Von: Vienna, Abflugszeit: Thu Jan 15 07:56:07 1970
Nach: Munich, Ankunftszeit: Thu Jan 15 07:56:07 1970
Dauer: 0h 31m

Flugnummer: 1246
Von: Munich, Abflugszeit: Thu Jan 15 11:14:28 1970
Nach: Amsterdam, Ankunftszeit: Thu Jan 15 11:14:28 1970
Dauer: 2h 46m

Beispiel 2: Stücklistenverwaltung

Lösungsidee:

Der hier beschriebene Code setzt sich aus verschiedenen Klassen und deren Beziehungen zueinander zusammen. Die Basisklasse `Part` dient als Grundlage für alle weiteren Klassen und enthält eine Member-Variable für den Namen des Teils sowie Methoden zum Abfragen des Namens und zum Vergleichen von Teilen miteinander.

Die Klasse `CompositePart` erbt von der Klasse `Part` und stellt ein zusammengesetztes Teil dar, das aus mehreren anderen Teilen bestehen kann. Sie enthält daher ein Vector von Pointern auf `Part`-Objekte, um die enthaltenen Teile zu speichern, sowie Methoden zum Hinzufügen von Teilen und zum Abfragen der enthaltenen Teile.

Die Klasse `Formatter` stellt eine Basisklasse für verschiedene Formatter dar, in denen Teile-Listen ausgegeben werden können. Sie enthält eine pure virtuelle Methode `printParts`, die von abgeleiteten Klassen implementiert werden muss.

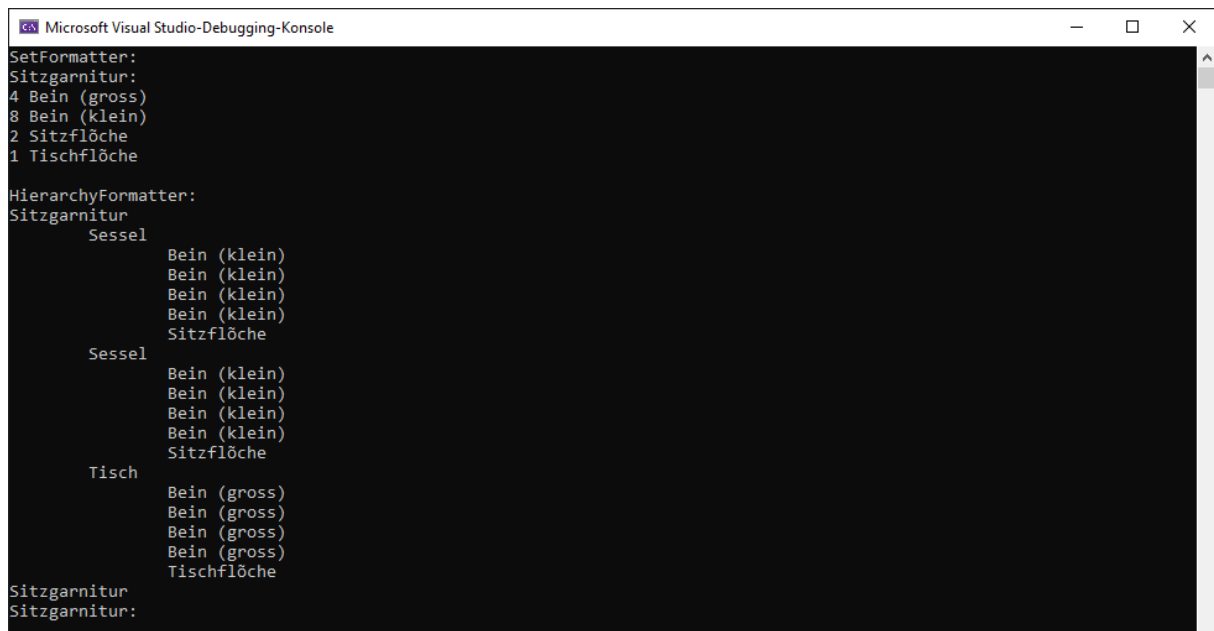
Die Klasse `SetFormatter` erbt von `Formatter` und implementiert die `printParts`-Methode in der Weise, dass sie alle Teile einer Teile-Liste in Form einer Set-Liste ausgibt, d.h. jedes Teil wird einmal aufgelistet, unabhängig davon, wie oft es in der Liste vorkommt.

Die Klasse `HierarchyFormatter` erbt ebenfalls von `Formatter` und implementiert die `printParts`-Methode in der Weise, dass sie die Hierarchie der Teile-Liste darstellt, indem sie Teile, die Teil von anderen Teilen sind, eingerückt ausgibt.

Die Funktionen `printParts` werden verwendet, um die Teile einer Stückliste auszugeben. Die Funktionen nehmen ein Objekt der Klasse `CompositePart` als Eingabeparameter und rufen für dieses Objekt und alle seine Teile die entsprechende Ausgabefunktion auf. Sie dienen als Wrapper Funktionen und rufen indirekt rekursive Funktionen auf, um auch alle unter-Teile auszugeben.

Testfälle:

```
5 // Erstelle eine Sitzgarnitur
6 void test1() {
7     CompositePart sitzgarnitur("Sitzgarnitur");
8
9     // Erstelle zwei Sessel und füge sie der Sitzgarnitur hinzu
10    CompositePart* sessel1 = new CompositePart("Sessel");
11    CompositePart* sessel2 = new CompositePart("Sessel");
12    sitzgarnitur.addPart(sessel1);
13    sitzgarnitur.addPart(sessel2);
14    // Erstelle vier kleine Beine und füge sie den Sesseln hinzu
15    Part* bein1 = new Part("Bein (klein)");
16    Part* bein2 = new Part("Bein (klein)");
17    Part* bein3 = new Part("Bein (klein)");
18    Part* bein4 = new Part("Bein (klein)");
19    sessel1->addPart(bein1);
20    sessel1->addPart(bein2);
21    sessel1->addPart(bein3);
22    sessel1->addPart(bein4);
23    sessel2->addPart(bein1);
24    sessel2->addPart(bein2);
25    sessel2->addPart(bein3);
26    sessel2->addPart(bein4);
27
28    // Erstelle eine Sitzfläche und füge sie den Sesseln hinzu
29    Part* sitzflaeche = new Part("Sitzfläche");
30    sessel1->addPart(sitzflaeche);
31    sessel2->addPart(sitzflaeche);
32
33    // Erstelle einen Tisch und füge ihn der Sitzgarnitur hinzu
34    CompositePart* tisch = new CompositePart("Tisch");
35    sitzgarnitur.addPart(tisch);
36
37    // Erstelle vier grosse Beine und füge sie dem Tisch hinzu
38    Part* bein5 = new Part("Bein (gross)");
39    Part* bein6 = new Part("Bein (gross)");
40    Part* bein7 = new Part("Bein (gross)");
41    Part* bein8 = new Part("Bein (gross)");
42    tisch->addPart(bein5);
43    tisch->addPart(bein6);
44    tisch->addPart(bein7);
45    tisch->addPart(bein8);
46
47    // Erstelle eine Tischfläche und füge sie dem Tisch hinzu
48    Part* tischflaeche = new Part("Tischfläche");
49    tisch->addPart(tischflaeche);
50
51    // Erstelle einen SetFormatter und einen HierarchyFormatter
52    SetFormatter set_formatter;
53    HierarchyFormatter hierarchy_formatter;
54
55    // Gib die Stückliste mit dem SetFormatter aus
56    std::cout << "SetFormatter:\n";
57    set_formatter.printParts(sitzgarnitur);
58
59    // Gib die Stückliste mit dem HierarchyFormatter aus
60    std::cout << "\nHierarchyFormatter:\n";
61    hierarchy_formatter.printParts(sitzgarnitur);
62
63    // Öffne eine Datei zum Schreiben
64    std::ofstream out("sitzgruppe.txt");
65
66    // Speichere die Sitzgruppe in der Datei
67    sitzgarnitur.store(out);
68
69    // Schliesse die Datei
70    out.close();
71
72    // Öffne die Datei zum Lesen
73    std::ifstream in("sitzgruppe.txt");
74
75    // Lade die Sitzgruppe aus der Datei
76    auto* sitzgarnitur_kopie = CompositePart::load(in);
77
78    // Schliesse die Datei
79    in.close();
80
81    hierarchy_formatter.printParts(*sitzgarnitur_kopie);
82    set_formatter.printParts(*sitzgarnitur_kopie);
83
84 }
```



```
Microsoft Visual Studio-Debugging-Konsole

SetFormatter:
Sitzgarnitur:
4 Bein (gross)
8 Bein (klein)
2 Sitzfläche
1 Tischfläche

HierarchyFormatter:
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
Sitzgarnitur:
```

Sitzgruppe.txt

```
1 Sitzgarnitur
2 Sessel
3 Bein (klein)
4 Bein (klein)
5 Bein (klein)
6 Bein (klein)
7 Sitzfläche
8 Sessel
9 Bein (klein)
10 Bein (klein)
11 Bein (klein)
12 Bein (klein)
13 Sitzfläche
14 Tisch
15 Bein (gross)
16 Bein (gross)
17 Bein (gross)
18 Bein (gross)
19 Tischfläche
```

Anmerkung: Das Speichern der Waren in einem File funktioniert grundsätzlich, nur das Laden nicht wirklich da beim speichern die Hierarchie verloren geht. Aufgrund Zeitmangels konnte diese Funktion aber nicht mehr vollständig implementiert werden.

Zeitaufwand: ca. 9h