Name: Theresa Gschweidl
Aufwand in h: 20 h

Übung zu Klassen und Vererbung

Inhalt

Beispiel 1: Flugreisen 2
Lösungsidee 2
Testfälle 2
Testfall 1 – Klasse person: 2
Testfall 2 – Klasse customer: 3
Testfall 3 – Klasse Flight: 3
Testfall 4 – Klasse AirTravel: 3
Testfall 5 – Flugreise Beispiel: 3
Beispiel 2: Eine Herakles Aufgabe – Stücklistenverwaltung 6
Lösungsidee 6
Testfälle 7
Testfall equals 7
Testfall CompositeParts und HierarchyFormatter 7
Testfall CompositePart und SetFormatter 8
Testfall diverse: 9
Testfall load from empty file 10
Testfall load and store 11

Beispiel 1: Flugreisen

Lösungsidee

In der Klasse Person werden alle Informationen zu einem Kunden gespeichert (das sind Vor- und Nachname, Geburtsdatum & Alter, Adresse, Kreditkartennummer und das Geschlecht).

In der Klasse Flight sind alle Details zu einem einzelnen Flug (ohne Umsteigen) zwischen zwei Städten gespeichert. Das heißt Ankunfts- und Abflugsort, die jeweiligen Uhrzeiten, die Flugnummer, die Airline und die Flugdauer.

Die Klasse AirTravel fasst eine Reise zusammen, das heißt hier sind alle während einer Reise getätigten Flüge in einem Vektor gespeichert. Außerdem gibt es einen eigenen Vektor, welche die Destinationen speichert (z.b. sind Städte in denen nur umgestiegen wird keine Destinationen) und eine Passengerlist, in welcher alle an der Reise teilnehmenden Kunden gespeichert sind (z.B. für Reisegruppen, Paare oder Familien, die eine Reise gemeinsam buchen).

Der in AirTravel überladene <<Operator ermöglicht die Ausgabe der einzelnen Reiseabschnitte und deren Zwischenflughäfen, z.B. Hinflug: Wien -> Zürich -> London, Abschnitt 2: London -> New York, Abschnitt 3: New York -> Washington, Rückflug: Washington -> New York -> Zürich -> Wien

In der Klasse Flight gibt es eine Methode PrintItinerary, welche alle Informationen zu einem bestimmten Flug ausgibt (z.b. Wien -> Zürich mit Flugnummer, Airline, Dauer, Abflug- und Ankunftszeit).

Die Klasse Person verfügt über eine Methode, welche alle Details zu einem Kunden ausgibt, also die Kundenakte druckt.

Testfälle

Testfall 1 – Klasse person:

Ziel: Person und zugehörige Informationen speichern und Kundenfile ausgeben:

```
Testing class person and printing customer file:
Name: Clark Kent (m)
Age: 30
Address: Favoritenstrasse 34, 1100 Vienna
Credit Card Number: 253422916


--------------------------------
Name: Lois Lane (f)
Age: 29
Address: Favoritenstrasse 34, 1100 Vienna
Credit Card Number: 532185455
```

Testfall 2 – Klasse customer:

```
Testing class customers and printing customer files for vienna customers:
0: Name: Clark Kent (m)
Age: 30
Address: Favoritenstrasse 34, 1100 Vienna
Credit Card Number: 253422916

1: Name: Lois Lane (f)
Age: 29
Address: Favoritenstrasse 34, 1100 Vienna
Credit Card Number: 532185455
```

Testfall 3 – Klasse Flight:

Flugdetails und Flugplan für eine Einzelflug ausgeben:

```
Testing class flight and print itinerary:
Airline: Lufthansa
Flightnumber: LH 6933
-----------------------
Linz --> Frankfurt
Take Off: 14:25
Arrival time: 15:30
Flight duration: 1hrs 5min
```

Testfall 4 – Klasse AirTravel:

Flugreisen mit mehr Stops, layovers und ohne Round-Trip:

```
Testing AirTravel Journey - 1: Roundtrip USA with one Destination stops:
Flight outbound: Linz --> Frankfurt --> Denver --> Las Vegas
Flight inbound: Las Vegas --> San Francisco --> Munich --> Linz

Testing AirTravel Journey - 2: Roundtrip USA with two Destination stops:
Flight outbound: Linz --> Frankfurt --> Denver --> Las Vegas
Travel section 2: Las Vegas --> San Francisco
Flight inbound: San Francisco --> Munich --> Linz

Testing AirTravel Journey - 3: Simple Trip with no return:
Flight outbound: Vienna --> London

Testing AirTravel Journey - 4: Simple Round Trip with no layovers:
Flight outbound: Vienna --> London
Flight inbound: London --> Vienna
```

Testfall 5 – Flugreise Beispiel:

Print Travel Itineraries:

```
Print Complete Itinerary incl. Passenger Details for europe-flight:
Travel Information for:
Name: Natascha Romanoff (f)
Age: 32
Address: Universal City Plaza 100, 91608 Universal City
Credit Card Number: 1234568291
Name: Clint Barton (m)
Age: 33
Address: Universal City Plaza 100, 91608 Universal City
Credit Card Number: 1239987876
------------------------------------------------------

Complete Itinerary:
----------
Airline: Austrian Airlines
Flightnumber: OS 3478
------------------------
Vienna --> Zurich
Take Off: 10:20
Arrival time: 11:40
Flight duration: 1hrs 20min
-----------------------------------
Airline: Swiss
Flightnumber: CH 3342
------------------------
Zurich --> London City Airport
Take Off: 12:20
Arrival time: 14:40
Flight duration: 2hrs 20min
-----------------------------------
```

```
Print Complete Itinerary incl. Passenger Details for US-West-flight:
Travel Information for:
Name: Lucy Chen (f)
Age: 30
Address: Wilshire Blvd 8850, 90211 Beverly Hills
Credit Card Number: 191282168
Name: Timothy Bradford (m)
Age: 34
Address: Wilshire Blvd 8850, 90211 Beverly Hills
Credit Card Number: 191282168
------------------------------------------------------

Complete Itinerary:
----------
Airline: United
Flightnumber: UA 2381
------------------------
Los Angeles --> Las Vegas
Take Off: 10:20
Arrival time: 11:40
Flight duration: 1hrs 20min
-----------------------------------
```

```
---------------------------------
Print Short Itinerary with Operator<<:
Flight outbound: Vienna --> Zurich --> London City Airport

Flight outbound: Los Angeles --> Las Vegas

-------------------------------------------------
```

Print all Customers on File for Hollywood Division:

```
-------------------------------------------------------
Print all Customers on File for Hollywood Division:
0: Name: Natascha Romanoff (f)
Age: 32
Address: Universal City Plaza 100, 91608 Universal City
Credit Card Number: 1234568291

1: Name: Clint Barton (m)
Age: 33
Address: Universal City Plaza 100, 91608 Universal City
Credit Card Number: 1239987876

2: Name: Tony Stark (m)
Age: 42
Address: Malibu Drive 10, 90263 Malibu
Credit Card Number: 1981276374

3: Name: Lucy Chen (f)
Age: 30
Address: Wilshire Blvd 8850, 90211 Beverly Hills
Credit Card Number: 191282168

4: Name: Timothy Bradford (m)
Age: 34
Address: Wilshire Blvd 8850, 90211 Beverly Hills
Credit Card Number: 191282168
```

```
Customers hollywood_office;
person::Person p1;
p1.setPerson("Natascha", "Romanoff", 1990, 05, 12, 32, 1234568291, person::female, "Universal
hollywood_office.AddPerson(&p1);
person::Person p2;
p2.setPerson("Clint", "Barton", 1989, 05, 20, 33, 1239987876, person::male, "Universal City Pl
hollywood_office.AddPerson(&p2);
person::Person p3;
p3.setPerson("Tony", "Stark", 1980, 10, 10, 42, 1981276374, person::male, "Malibu Drive", 10,
hollywood_office.AddPerson(&p3);
person::Person p4;
p4.setPerson("Lucy", "Chen", 1992, 10, 01, 30, 191282168, person::female, "Wilshire Blvd", 885
hollywood_office.AddPerson(&p4);
person::Person p5;
p5.setPerson("Timothy", "Bradford", 1988, 10, 01, 34, 191282168, person::male, "Wilshire Blvd"
hollywood_office.AddPerson(&p5);
```

Beispiel 2: Eine Herakles Aufgabe – Stücklistenverwaltung

Lösungsidee

Im Namensraum PartsLists gibt es die abstrakte **Klasse Storable**, welche das Interface zu Speicherung im Filesystem und laden vom Filesystem zur Verfügung stellt. Die Basisklasse für die eigentliche Datenstruktur der Stückliste ist die **Klasse Parts**. Davon abgeleitet gibt es die **Klasse CompositeParts**.

In der **Klasse Parts** gibt es einen Constructor, welchem der Name des Teils bereits bei der Initialisierung übergeben werden muss. Für den Zugriff auf den Namen gibt es die Methode getName() → diese retourniert den als private Datenkomponente angelegten Namen. Für die leichtere Handhabung beim Einlesen vom File, habe ich ebenfalls eine Methode mit setName() erstellt, um den Namen später gegebenenfalls ändern zu können.

Mit der Methode equals können zwei Teile verglichen werden, das heißt die Methode gleicht die Namen ab, ob dieser gleich ist. Ich habe diese nicht gebraucht und wusste auch nicht, wo ich sie speziell einbauen sollte. Desweiteren wurden noch die beiden aus dem Storable Interface übernommenen Methoden load und store für Part implementiert. Load wird in CompositePart überschrieben, da die load-Methode von Part nur einen einzelnen atomaren Teil einlesen kann.

Load

Load liest zeilenweise von einem File ein und speichert jede Zeile als Element eines Vektors. Das File-Format wurde so gewählt, dass beim HierarchyFormatter das erste Element jeder Zeile angibt, ob es sich um einen atomaren Part handelt oder einen CompositePart.
Nachdem das File fertig eingelesen wurde:

- Durch den Vektor iterieren und je nach dem um welchen Teil es sich handelt:
    - Bei CompositeParts wird ein neuer CompositePart erstellt (mit Pointer und Schlüsselword new → Anmerkung: bin mir nicht sicher wo genau, dass das delete gehören würde. Mit „delete this“ in destruktoren hat das gesamte Programm nicht mehr funktioniert).
    - Bei atomaren Parts wird ein neuer Part erstellt und dieser anschließend beim letzten erstellten CompositePart angehängt.
    - Wenn ein neues CompositePart erstellt werden muss UND die Iteration durch den Vektor das erste Elemente bereits hinter sich gelassen hat wird das aktuelle CompositePart an das this-Objekt angehängt (mit addPart) und erst danach der Pointer mit einem neuen CompositePart überschrieben.
- Letztes CompositePart Element aus dem Vektor an this-Objekt anfügen

Die **Klasse CompositePart** erst das Attribut Name von der Basisklasse. Der eigene Konstruktor ruft für die Namenszuweisung den Konstruktor der Basisklasse auf. Die Methode addPart ist nur in der abgeleiteten Klasse verfügbar, da diese auch über einen Vector, welcher alle atomaren Teile zusammenfasst, verfügt. Mit getParts() kann der Vektor aufgerufen werden (dieser ist privat und sonst nicht accessible). Die Methode load wird hier für CompositeParts neu implementiert (siehe oben).

Die beiden Formatter Klassen ermöglichen die Ausgabe der Stückliste und unterschiedlichem Format. Die **Klasse HierarchyFormatter** beinhaltet eine rekursive Funktion, welche durch die

Beziehungshierarchie des aufrufenden Objektes iteriert und entsprechend ausgibt. Je tiefer die Rekursion, desto höher der Einzug von rechts. In der **Klasse SetFormatter** gibt es eine Map als private Datenkomponente. Mit Hilfe dieser, werden alle Einzelteile als Schlüssel gespeichert und bei mehrmaligen Vorkommen der dazugehörige Wert entsprechend erhöht. Die Map wird allerdings in einer sparaten privaten Methode erstellt, damit bei den rekursiven Aufrufen nicht mehrmals gezählt bzw. gedruckt wird. Für die Ausgabe wird mithilfe eines Iterators durch die Map iteriert und die jeweiligen Key-Value Pairs ausgegeben.

Testfälle

Testfall equals

Test erfolgreich:

```
Testing method equals():
Test part 1:
Input parameters: leg1 = Tischbein leg2 = Stuhlbein
Tischbein is not equal to Stuhlbein

-----------------------------------
Test part 2:
Input parameters: leg1 = Tischbein leg3 = Tischbein
Tischbein is equal to Tischbein
```

```
testequals() {
td::cout << "Testing method equals():\n";
art leg1("Tischbein");
art leg2("Stuhlbein");
art leg3("Tischbein");
td::cout << "Test part 1: \n";
td::cout << "Input parameters: leg1 = " << leg1.getName() << " leg2 = " << leg2.getName() << '\n';
f (leg1.equals(leg2)) {
    std::cout << leg1.getName() << " is equal to " << leg2.getName() << '\n';

lse {
    std::cout << leg1.getName() << " is not eq to " << leg2.getName() << '\n';


td::cout << "\n-----------------------------------\n";
td::cout << "Test part 2: \n";
td::cout << "Input parameters: leg1 = " << leg1.getName() << " leg3 = " << leg3.getName() << '\n';
f (leg1.equals(leg3)) {
    std::cout << leg1.getName() << " is equal to " << leg3.getName() << '\n';

lse {
    std::cout << leg1.getName() << " is not equal to " << leg3.getName() << '\n';
```

Testfall CompositeParts und HierarchyFormatter

Test erfoglreich

```
Testing CompositePart List and HierarchyFormatter:
Print with HierarchyFormatter:
C: Wohnkueche
C:   Sitzecke
C:     Tisch
P:       Tischbein
P:       Tischbein
P:       Tischbein
P:       Tischbein
P:       Tischplatte
C:   Kueche
P:     Mikrowelle
P:     Kuehlschrank
C:     Kuecheninsel
P:       Herd
C:       Moebelstueck
P:         Lade
P:         Lade
P:         Lade
```

```
void testCompositePart() {
    std::cout << "Testing CompositePart List and HierarchyFormatter:\n";
    CompositePart livingspace("Wohnkueche");
    CompositePart sitzecke("Sitzecke");
    CompositePart table("Tisch");
    Part leg("Tischbein");
    Part tabletop("Tischplatte");
    table.addPart(&leg);
    table.addPart(&leg);
    table.addPart(&leg);
    table.addPart(&leg);
    table.addPart(&tabletop);
    sitzecke.addPart(&table);
    livingspace.addPart(&sitzecke);
    CompositePart kitchen("Kueche");
    Part microwave("Mikrowelle");
```

```
Part microwave("Mikrowelle");
kitchen.addPart(&microwave);
Part fridge("Kuehlschrank");
kitchen.addPart(&fridge);
CompositePart island("Kuecheninsel");
Part oven("Herd");
CompositePart storage("Moebelstueck");
Part drawer("Lade");
storage.addPart(&drawer);
storage.addPart(&drawer);
storage.addPart(&drawer);
island.addPart(&oven);
island.addPart(&storage);
kitchen.addPart(&island);
livingspace.addPart(&kitchen);
```

Testfall CompositePart und SetFormatter

Test erfoglreich:

```
    SetFormatter set;
    std::cout << "\n\nPrint with SetFormatter:\n";
    set.printParts(livingspace);
}
```

```
Print with SetFormatter:
Wohnkueche
    Herd: 1
    Kueche: 1
    Kuecheninsel: 1
    Kuehlschrank: 1
    Lade: 3
    Mikrowelle: 1
    Moebelstueck: 1
    Sitzecke: 1
    Tisch: 1
    Tischbein: 4
    Tischplatte: 1
```

Testfall diverse:

Test: Part und CompositePart anlegen und Namen ausgeben:

```
void testPart() {
    std::cout << "Test Part:\n";
    Part tablet("tabletop");
    std::cout << tablet.getName();
    std::cout << "\n---------------------------\n\n";

    Part tableleg("table leg");
    CompositePart table("table");
    std::cout << "Test CompositePart:\n";
    std::cout << table.getName();
```

```
Test Part:
tabletop
---------------------------

Test CompositePart:
table

---------------------------
```

Test SetFormatter:

```
    std::cout << "\n\nTest SetFormatter:\n";
    HierarchyFormatter hier;
    SetFormatter set;
    set.printParts(table);
    std::cout << "\n------\n";
    set.printParts(sofaset);
    std::cout << "\n---------------------------\n\n";
```

```
Test SetFormatter:
table
   table leg: 2
   tabletop: 1

------
Sofaset
   leg: 1
   sofachair: 1
   table: 1
   table leg: 4
   tabletop: 2

---------------------------
```

Test HierarchyFormatter:

```
 std::cout << "\nTest HierarchyFormatter:\n";
 hier.printParts(sofaset);
```

```
Test HierarchyFormatter:
C: Sofaset
C:   table
P:     tabletop
P:     table leg
P:     table leg
C:   sofachair
P:     leg
```

Testfall load from empty file

```
Test load from empty file. Test Oject to call load has been initialized with name test1
test1
C:\Users\there\Documents\SWE3UE\SWE3_Uebung_05_bis_28.12.2022\SWE3_Gschweidl_Ue05\x64\Deb
```

```
void testEmptyFile() {
    std::ifstream inFile ("empty.txt");
    Part test1("test1");
    test1.load(inFile);
    SetFormatter set;
    set.printParts(test1);
    std::cout << "Test load from empty file. Test Oject to call load has been initialized wi
    std::cout << test1.getName();
}
```

→Test erfolgreich, da erwartetes Verhalten.

Testfall load and store

```
Test load and store with sofaset:
Test store to file --> result see file

C:\Users\there\Documents\SWE3UE\SWE3_Uebung 05_bis 28.12.2022\SWE3_Gschweidl_Ue05\x64\Debug\Ue05_Bsp2.exe (Prozess "3527
2") wurde mit Code "3" beendet.
Drücken Sie eine beliebige Taste, um dieses Fenster zu schließen.
```

```
>Partslist for Sofaset
C: Sofaset
C:   table
P:     leg
P:     leg
P:     leg
P:     leg
P:     tabletop
C:   sofachair
P:     seatcushion
P:     leg
P:     leg
P:     leg
P:     leg
```

→Store to file ist erfolgreich, load from file nicht → out of Range Exception

```
    sofaset.addPart(&sofachair);

    std::ofstream outFile("output.txt");
    sofaset.store(outFile);
    outFile.close();
    std::cout << "Test store to file --> result see file\n";

    std::ifstream inFile("output.txt");
    CompositePart inObject("tmp");
    inObject.load(inFile);

    std::cout << "Test load from File:\n";
    HierarchyFormatter hiera;
    hiera.printParts(inObject);
```

Update → OutOfRange Error fixed:


Test load and store with sofaset:
Test store to file --> result see file
Test load from File:
C: Sofaset
C: table
P: leg
P: leg
P: leg
P: leg
P: tabletop
C: sofachair
P: seatcushion
P: leg
P: leg
P: leg
P: leg
C:\Users\there\Documents\SWE3UE\SWE3 Uebung