h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Programmieren, Algorithmen und Datenstrukturen II

Praktikum 2: Vererbung und Komposition

Sommersemester 2019

Prof. Dr. Arnim Malcherek

Allgemeine Hinweise zum Praktikum:

- Bereiten Sie die Aufgaben unbedingt zu Hause oder in einem freien Labor vor. Das beinhaltet:
  - Entwurf der Lösung
  - Codieren der Lösung in einem Qt-Creator-Projekt
- Die Zeit während des Praktikums dient dazu, die Lösung testieren zu lassen sowie eventuelle Korrekturen vorzunehmen.
- Das Praktikum dient auch zur Vorbereitung der praktischen Klausur am Ende des Semesters. Versuchen Sie also in Ihrem eigenen Interesse, die Aufgaben selbständig nur mit Verwendung Ihrer Unterlagen bzw. Ihres bevorzugten C++-Buches und ohne Codefragmente aus dem Netz zu lösen.
- Die Lösung wird nur dann testiert, wenn
  - sie erklärt werden kann bzw. Fragen zur Lösung beantwortet werden können.
  - das Programm ablauffähig und die Lösung nachvollziehbar ist.
  - die Hinweise oder Einschränkungen aus der Aufgabenstellung befolgt wurden.
- Zur Erinnerung hier noch einmal die Regeln des Praktikums, die schon in der Vorlesung besprochen wurden:
  - Sie arbeiten in 2er Gruppen.
  - Ein Testat gibt es nur zum jeweiligen Termin.
  - Abschreiben und Kopieren ist verboten.
  - Es gibt keine Noten. Die Bewertung ist lediglich erfolgreich / nicht erfolgreich.
  - Das Praktikum ist Zulassungsvoraussetzung für die Klausur. Hierfür müssen alle fünf Praktikumsübungen testiert sein.

Lernziele:

- Sie lernen Vererbung im Softwareentwurf und in der Implementierung einzusetzen.
- Sie kennen die verschiedenen Beziehungen zwischen Klassen.

In diesem Praktikum bauen wir auf den Ergebnissen des letzten Praktikums auf und erweitern unsere Klassenstrukturen.

Aufgabe 1

Implementieren Sie das Klassendiagramm aus Abbildung 1. Ihr Coding aus dem letzten Praktikum können Sie natürlich als Ausgangspunkt verwenden. Die Attribute `id`, `price`,

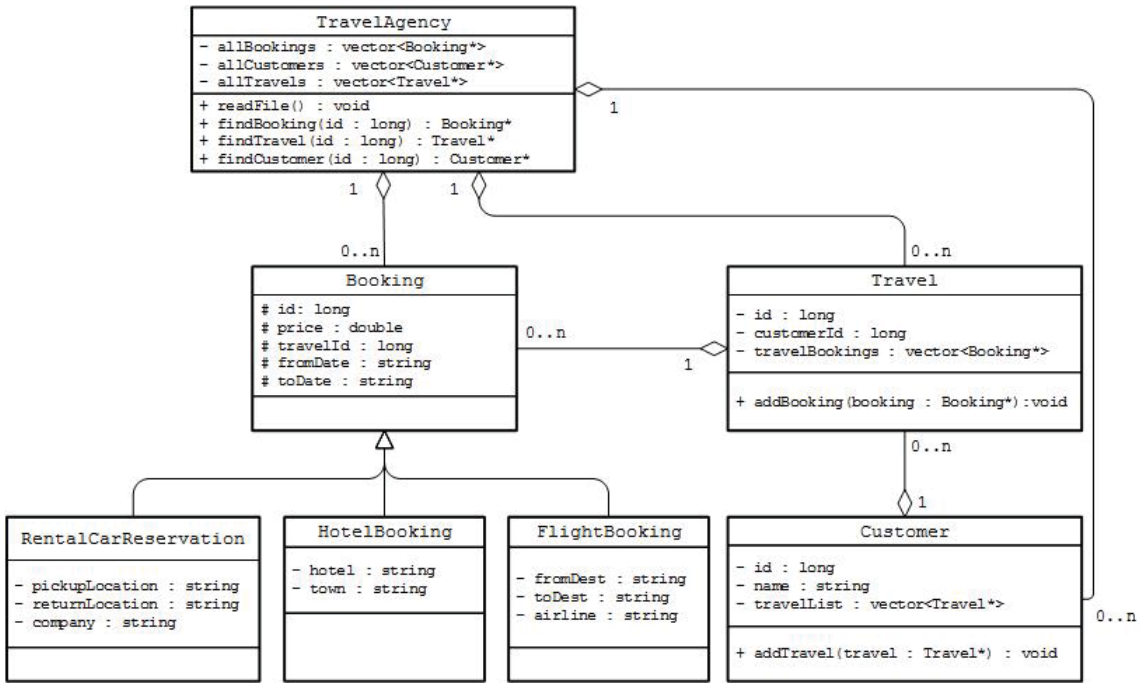

Abbildung 1: Klassendiagramm

`travelId`, `fromDate` und `toDate` sind allen Buchungen gemeinsam und liegen darum in der Basisklasse `Booking`.

Aufgabe 2

Das Format der Datei bookings2.txt hat sich jetzt leicht verändert. Es gibt zusätzliche Daten:

Auszug:

```
F|35|277.01|20150602|20150603|7|4|Uli Stein|STR|WAW|Lufthansa
R|36|167.01|20150602|20150603|7|4|Uli Stein|CHOPIN FLUGH.|CHOPIN FLUGH.|Europcar
H|37|256.02|20150602|20150603|7|4|Uli Stein|Residence 1898|Warschau
F|38|288.93|20150603|20150603|7|4|Uli Stein|WAW|STR|Lufthansa
```

Spalte 1 beschreibt den Typ der Buchung. Die Spalten 9-11 hängen vom Typ ab. Hinzugekommen sind im Vergleich zum letzten Praktikum lediglich die Spalten 6-8.

Spalte 1:	Typ der Buchung (`F` für Flight, `H` für Hotel, `R` für Rental Car)
Spalte 2:	ID der Buchung (Attribut `id`)
Spalte 3:	Preis der Buchung (Attribut `price`)
Spalte 4:	Start-Datum der Buchung im Format JJJJMMTT (Attribut `fromDate`)
Spalte 5:	End-Datum der Buchung im Format JJJJMMTT (Attribut `toDate`)
Spalte 6:	ID der Reise (Booking::travelId und Travel::id)
Spalte 7:	ID des Kunden (Customer::id und Travel::customerId)
Spalte 8:	Name des Kunden (Customer::name)
Spalte 9:	falls Typ = 'F': Startflughafen (`FlightBooking::fromDest`)
	falls Typ = 'R': Abholstation des Autos (`RentalCarReservation::pickupLocation`)
	falls Typ = 'H': Hotelname (`HotelBooking::hotel`)
Spalte 10:	falls Typ = 'F': Zielflughafen (`FlightBooking::toDest`)
	falls Typ = 'R': Rückgabestation des Autos (`RentalCarReservation::returnLocation`)
	falls Typ = 'H': Name der Stadt (`HotelBooking::town`)
Spalte 11:	falls Typ = 'F': Fluglinie (`FlightBooking::airline`)
	falls Typ = 'R': Autoverleihfirma (`RentalCarReservation::company`)

Lesen Sie die Datei ein und legen Sie die Objekte zu den verschiedenen Klassen an. Natürlich müssen auch die Vektoren mit den enthaltenen Objekten gefüllt werden (z.B. `Travel::travelBookings`).
Wichtig: Jedes Objekt zu einer `id` wird nur einmal angelegt.
Beispiel: Obwohl es n Zeilen mit der `customerId` 1 gibt, darf der entsprechende Kunde nur einmal angelegt werden. Genauso für Reisen: Es kann viele Buchungen zu einer Reise geben, aber natürlich darf die Reise nur einmal angelegt werden. Im Unterschied zum ersten Praktikum landen jetzt alle Buchungen im `vector allBookings`, der Zeiger auf die Basisklasse enthält.
Wieso kann man hier auch Zeiger auf abgeleitete Klassen speichern?

Tipp: Eine einfache Methode ist, vor dem Anlegen eines Objektes mit der entsprechenden `find`-Methode (z.B. `findCustomer`) zu prüfen, ob es das Objekt schon gibt.

- Geben Sie zum Abschluss des Einlesens wieder eine Meldung aus, in der Sie mitteilen, wie viele Objekte der verschiedenen Klassen Sie angelegt haben. Geben Sie anschließend eine Liste aller Kunden und Reisen aus.
- Geben Sie zusätzlich aus, wie viele Reisen der Kunde mit der ID 1 gebucht hat, und wie viele Buchungen die Reise mit der ID 17 beinhaltet.

Beispielausgabe (Natürlich müssen Sie die Fragezeichen durch die korrekten Werte ersetzen):

```
Es wurden ? Flugreservierungen, ? Hotelbuchungen und ? Mietwagenreservierungen
```

```
im Gesamtwert von ? eingelesen. Es wurden ? Reisen und ? Kunden angelegt. Der
Kunde mit der ID 1 hat ? Reisen gebucht. Zur Reise mit der ID 17 gehören ? Buchun-
gen.
```

Aufgabe 3

Legen Sie in der Klasse `TravelAgency` eine Methode

```
int createBooking(  char type, double price, string start, string end,
                    long travelId, vector<string> bookingDetails)
```

an.

Die Aufgabe der Methode ist es, eine Buchung vom Typ `type` anzulegen. Die gemeinsamen Felder aller Booking Typen sind als einzelne Parameter in der Schnittstelle der Methode enthalten. Die speziellen Felder der einzelnen Buchungstypen sind im Vektor `bookingDetails` enthalten.

Die Methode muss zusätzlich kontrollieren, ob das Objekt `Travel` mit der entsprechenden `id` bereits existiert. Falls dieses Objekt nicht existiert, soll die Methode den Wert `-1` zurückgeben und keine Buchung anlegen. Die `id` der Buchung muss vor dem Anlegen ermittelt werden. Dazu soll die größte existierende `id` bestimmt werden. Die neue `id` soll dann gerade um `1` größer sein.

Nach erfolgreichem Anlegen der Buchung und Speicherung im `vector allBookings` soll die Methode die `id` der angelegten Buchung zurückgeben. Testen Sie die Methode durch einen Aufruf in `main` und geben Sie anschließend zur Kontrolle wieder eine Meldung wie im Beispiel unten aus, wo sie die neue Buchung jetzt mitzählen.

```
Es wurden ? Flugreservierungen, ? Hotelbuchungen und ? Mietwagenreservierungen
im Gesamtwert von ? eingelesen.
```