Objektorientierte Programmierung — Blatt 6

Institut für Softwaretechnik und Programmiersprachen | Sommersemester 2024
Matthias Tichy, Raphael Straub und Florian Sihler

Abgabe (git) bis
9. Juni 2024

Collections and Generics

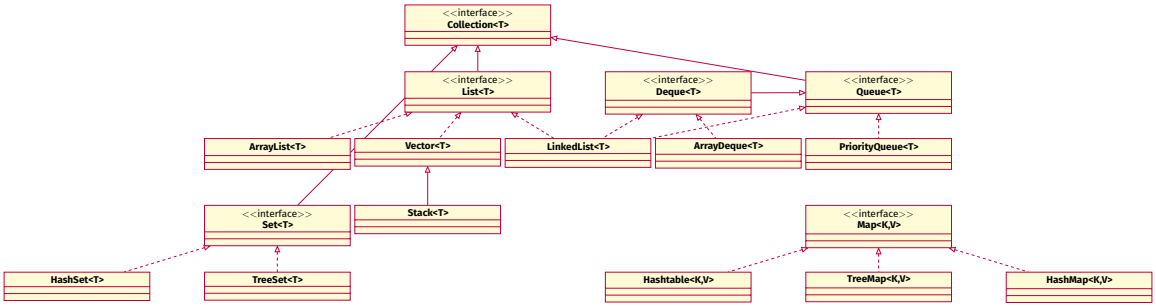
5 ★ 9 ■ 7 ●

- Generics verstehen und benutzen
- Erste Schritte mit Collections

Aufgabe 1: Choosing the Right Data Structure

In dieser Aufgabe möchten wir verschiedene Situationen betrachten, in denen es darauf ankommt, die richtige Datenstruktur für die jeweilige Anforderung zu wählen. In jeder Teilaufgabe wird eine Situation beschrieben, und Sie sollen entscheiden, welche Collection-Klasse am besten geeignet ist und warum. Wählen Sie zunächst das passende Collection-Interface und begründen Sie Ihre Wahl und dann die passende Implementierungsklasse (ebenfalls mit Begründung). Um die Wahl der Implementierungsklasse zu begründen, beachten Sie die unterschiedlichen (Laufzeit-)Komplexitäten der Operationen und die Anforderungen der jeweiligen Situation.

Zur Auswahl stehen die folgenden Collection-Interfaces und Implementierungsklassen:



a) Liste von Aufgaben ●

Sie programmieren eine To-Do-Listen-Anwendung. Die To-Do-List soll die Reihenfolge an Aufgaben beibehalten und das einfache Entfernen von erfüllten Aufgaben ermöglichen. Welche Datenstruktur würden Sie verwenden?

b) Einträge in einer Warteschlange ●

Stellen Sie sich vor, Sie programmieren ein System für die Verwaltung von Kunden in einem Call-Center. Die Kunden werden in der Reihenfolge ihrer Anrufe bedient. Wenn ein Kunde bedient wird, soll dieser aus der Warteschlange entfernt werden. Welche Datenstruktur würden Sie wählen?

c) Eindeutige Benutzernamen 🟢

Sie arbeiten an einer Social-Media-Plattform und müssen sicherstellen, dass alle gespeicherten Benutzernamen eindeutig sind. Welche Datenstruktur würden Sie verwenden?

d) Häufigkeit von Wörtern 🟢

Sie entwickeln ein Textanalyse-Tool, das die Häufigkeit von Wörtern in einem Dokument zählt. Welche Datenstruktur würden Sie wählen?

Aufgabe 2: Student Management System

In dieser Aufgabe werden Sie ein einfaches Student Management System implementieren, welches die Verwaltung von Studierenden und deren Einschreibungen in Kurse ermöglicht. Für diese Aufgabe haben wir eine `Student` Klasse und eine `Course` Klasse vorgegeben, die Sie im vorbereiteten Repository finden.

a) Enroll Student ⭐🟦

Erweitern Sie die Student Klasse um eine Methode `enrollInCourse(Course course)` die einen Studierenden in einen Kurs einschreibt. Es steht Ihnen frei, die Daten zu speichern, wie Sie es für richtig halten.

b) Leave Course 🟦

Erweitern Sie die Student Klasse um eine Methode `leaveCourse(Course course)` die einen Studierenden aus einem Kurs ausschreibt.

c) Enroll Course ⭐🟦🟦

Erweitern Sie nun die Course Klasse um eine weitere Methode `addToStudents(List<Student> students)` die eine Liste an Studierenden in einen Kurs einschreibt. Hierfür müssen Sie, wie in Aufgabe a, eine geeignete Datenstruktur wählen und die Studierenden speichern.

d) Kick Students 🟦

Erweitern Sie die Course Klasse um eine Methode `removeStudents(List<Student> students)` die eine Liste an Studierenden aus einem Kurs entfernt.

e) Consistency ⭐🟦🟦🟦🟢

Man ist nun in der Lage, sich über das Student Objekt und über das Course Objekt in einen Kurs einzuschreiben. Beide Klassen benötigen die Information über die Einschreibung. In der aktuellen Implementierung ist es möglich, dass ein Student in einem Kurs eingeschrieben ist, aber der Kurs nicht weiß, dass der Student sich eingeschrieben hat. Um die Konsistenz zu gewährleisten, müssen Sie die Implementierung anpassen. Wenn ein Student sich einschreibt oder austritt, muss der Kurs darüber informiert werden und umgekehrt.

f) Main Class

Erstellen Sie eine `Main` Klasse mit einer `main` Methode, um die Funktionalität zu demonstrieren. Fügen Sie einige Studierende und Kurse hinzu, schreiben Sie sie ein und zeigen Sie die Ergebnisse.

Aufgabe 3: Separate by List Type

In dieser Aufgabe wollen wir eine Methode implementieren, die eine Liste bearbeitet. Wir wollen verschiedene Methoden für verschiedene Typen von Listen haben. Dafür wollen wir das Overloading Konzept benutzen und haben folgenden Code implementiert:

```
public void example(List<Integer> integerList) {

}

public void example(List<String> stringList) {

}
```

Leider kompiliert der Code nicht! Finden Sie heraus warum und korrigieren Sie den Code.

Tipp: Dies ist ein fundamentales Problem an Java. Der korrigierte Code ist daher eher ein Workaround als eine elegante Lösung.

Aufgabe 4: Advanced Airline Reservation System

In dieser Aufgabe werden Sie ein erweitertes Flugreservierungssystem implementieren, welches die Verwaltung von Flügen und Reservierungen ermöglicht, sowie die Möglichkeit bietet, verschiedene Berichte zu generieren und komplexe Operationen durchzuführen.

Für diese Aufgabe haben wir eine `Flight` Klasse, eine `Passenger` Klasse, eine `Reservation` Klasse, und eine `FlightSchedule` Klasse vorgegeben, die Sie im vorbereiteten Repository finden.

Die gegebenen Klassen verwenden die `LocalDateTime` Klasse aus dem `java.time` Package. Informieren Sie sich zunächst in der Java Documentation über die Klasse und deren Verwendung.

a) FlightSchedule Data

Erweitern Sie die `FlightSchedule` Klasse, sodass diese in der Lage ist, die geplanten Flüge und Reservierungen zu speichern. Implementieren Sie hierzu die folgenden Methoden:

- `void addFlight(Flight flight)`
- `Flight getFlight(String flightNumber)`
- `Reservation makeReservation(String flightNumber, Passenger passenger)`

Verwenden Sie die aktuelle Zeit als Zeitstempel für die Reservierung.

b) Boarding

Nun soll es dem Passagier möglich sein, einen Flug zu betreten.

Implementieren Sie dafür eine Methode `boolean boardFlight(Passenger passenger)` in der Klasse `Flight`. Stellen Sie sicher, dass der Passagier nur dann einsteigen kann, wenn er eine gültige Reservierung für den Flug hat. Um das zu ermöglichen, müssen Sie der Flight Klasse ermöglichen, auf die Reservierungen in der `FlightSchedule` Klasse zuzugreifen. Stellen Sie aus Datenschutzgründen sicher, dass die Flight Klasse nur Zugriff auf die Reservierungen des entsprechenden Flugs hat. Der Wert der Rückgabe soll `true` sein, wenn der Passagier erfolgreich eingestiegen ist, und `false`, wenn der Passagier nicht eingestiegen ist.

c) Missing Passengers 

Implementieren Sie in der Klasse `Flight` eine Methode

```
List<Passenger> getMissingPassengers()
```

die eine Liste der Passagiere zurückgibt, die eine Reservierung für den Flug haben, aber noch nicht eingestiegen sind. Passen Sie dafür das Datenmodell an, um alle notwendigen Informationen zur Verfügung zu haben.

d) Passengers Routes 

Implementieren Sie in der Klasse `FlightSchedule` die Methode

```
Map<Passenger, List<Flight>> getPassengerRoutes()
```

die eine Map zurückgibt, die jedem Passagier eine Liste von Flügen zuordnet, die er gebucht hat.

e) Validate Route 

Implementieren Sie in der Klasse `FlightSchedule` die Methode

```
boolean validateRoute(List<Flight> route)
```

die überprüft, ob die gegebene Liste von Flügen eine gültige Route darstellt. Der Startflughafen ist der Flug mit dem frühesten Abflugdatum und der Zielort ist der Flug mit dem spätesten Ankunftsdatum. Sie müssen sicherstellen, dass an jedem Ort, an dem der Passagier ankommt, auch ein gebuchter Flug abfliegt, der für den Passagier zeitlich erreichbar ist. Gehen Sie hierfür davon aus, dass der Passagier maximal eine Stunde benötigt, um zum nächsten Flug zu gelangen.

f) Flight Statistics 

Implementieren Sie in der Klasse `FlightSchedule` die Methode

```
Map<Flight, Integer> getFlightStatistics()
```

die eine Map zurückgibt, die jedem Flug die Anzahl der Passagiere zuordnet, die für diesen Flug eine Reservierung haben.

g) Passenger Statistics 

Implementieren Sie die Methode

```
Map<LocalDate, Integer> getDailyPassengerCount()
```

die eine Map zurückgibt, die jedem Datum die Anzahl der Passagiere zuordnet, die für einen Flug an diesem Tag eine Reservierung haben.

Zusätzlich, implementieren Sie die Methode

```
Map<String, Integer> getPassengerCountPerDestination()
```

die eine Map zurückgibt, die jedem Zielort die Anzahl der Passagiere zuordnet, die für diesen Zielort eine Reservierung haben.

h) Main Class

Erstellen Sie eine `Main` Klasse mit einer `main` Methode, um die Funktionalität zu demonstrieren.