Softwareentwicklung II (IB)

Blatt 5

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 28.01.2020 17:34

Inhaltsverzeichnis

Abgabe der Aufgabe auf diesem Blatt: bis 27.06.18, 08:00 Uhr durch Pushen in das zur Aufgabe gehörende GitHub-Repository.

Aufgabe 1 — Vorlesungen und Prüfungen

Das Repository für diese Aufgabe bekommen Sie unter https://classroom.github.com/a/RPVXoQuE.

Implementieren Sie folgende Klassen, Enums und Interfaces zur Verwaltung von Vorlesungen und Prüfungen:

`Group`

Die Enum `Group` repräsentiert die verschiedenen Studiengänge an der FK07:

- **IB**: Wirtschaftsinformatik (Bachelor)
- **IC**: Scientific Computing (Bachelor)
- **IF**: Informatik (Bachelor)
- **IG**: Informatik (Master)
- **IN**: Wirtschaftsinformatik (Master)
- **IS**: Stochastic Engineering (Master)

Die Enumwerte sind die Abkürzungen (IB, …) in der oben angegebenen Reihenfolge. Der Name, z.B. `Wirtschaftsinformatik`, wird in einer privaten Objektvariable `longname`. Außerdem gibt es eine private Objektvariable `bachelor`, die genau dann den Wert `true` hat, wenn es sich um einen Bachelorstudiengang handelt.

Die Enum enthält außerdem folgende Methoden:

```
public boolean isBachelor();
public boolean isMaster();
```

und redefiniert `toString`, so dass der Wert in der o.a. Form nach dem Doppelpunkt zurück gegeben wird.

Mit den beiden statischen Methoden

```
public static Set<Group> getBachelors();
public static Set<Group> getMasters();
```

kann die Menge der Enumwerte (nutzen Sie `values()`) berechnet werden, die Bachelor- bzw. Masterstudiengänge repräsentieren. Schreiben Sie in diese Methode auf keinen Fall Enumwerte als Literale direkt in die Menge! Die Methoden sollen auch dann noch richtig funktionieren, wenn wir in der Enum Enumwerte hinzufügen.

`Person`

Die Klasse `Person` enthält nur den Namen als Zeichenkette. Dazu einen Konstruktor, sowie die Methoden

```
public String getName();
@Override public String toString();
```

die beide den Namen zurück geben.

`Student`

Die Klasse `Student` erweitert die Klasse Person. Zusätzlich enthält Sie die Matrikelnummer als `int` mit einem Getter `getMatrikel`. Die Klasse soll die Matrikelnummer im Konstruktor als fortlaufende Nummer beginnend mit `1000000` vergeben. Verwenden Sie dazu eine private Klassenvariable und eine private statische Methode. Nutzen Sie im Konstruktor den Aufruf des Konstruktors von `Person`.

`Title`

Die Enum `Title` repräsentiert den Titel. Sie enthält folgende Werte: `Dr`, `Prof` und `ProfDr`. Die `toString`-Methode soll die Werte aber mit Punkten zurück geben, also `Dr.`, `Prof.` und `Prof. Dr.`. Nutzen Sie dazu eine Objektvariable.

`Lecturer`

Die Klasse `Lecturer` erweitert auch die Klasse `Person`. Sie enthält zusätzlich die private Objektvariable `title`, die den Titel des Dozenten enthält. Sie enthält die beiden Konstruktoren:

```
public Lecturer(final String name);
public Lecturer(final Title title, final String name);
```

Der erste Konstruktor ruft den Konstruktor von `Person` auf und lässt den Titel auf `null`. Der zweite Konstruktor ruft den ersten auf und setzt zusätzlich den Titel.

Für den Titel gibt es einen Setter.

Die redefinierte `toString`-Methode ruft entweder nur `toString` von `Person` auf oder, wenn der `Lecturer` einen Titel hat, wird der Titel (`toString` von `title`) gefolgt von einem Leerzeichen und toString`von`Person‘.

`Room`

Die Klasse `Room` enthält die beiden unveränderbaren Objektvariablen `name` und `seats` mit Gettern, die durch den Konstruktor gesetzt werden.

Die Klasse redefiniert sowohl `equals` als auch `hashCode`, so dass zwei Räume gleich sind, wenn Sie den gleichen Namen haben. Die Anzahl von Sitzpätzen ist für beide Methoden nicht relevant.

ExamRoom

Die Klasse `ExamRoom` erweitert die Klasse `Room` um eine unveränderliche Objektvariable `seatsForExam`, die die Sitzplätze, die für Prüfungen genutzt werden können, repräsentiert. Neben dem Getter `getSeatsForExam` hat die Klasse folgende Konstruktoren:

```
public ExamRoom(final String name, final int seats, final int seatsForExam);
public ExamRoom(final Room room, final int seatsForExam);
```

Rufen Sie aus dem ersten den Konstruktor von `Room` auf und aus dem zweiten den ersten.

Exam

Das Interface `Exam` enthält folgende Methoden:

```
int getDuration();
Lecturer getExaminer();
Set<Room> getExamRooms();
Set<Student> getStudents();
boolean validate();
```

für die Dauer, den Prüfer, die Prüfungsräume und die angemeldeten Studierenden sowie zur Validierung.

Examtype

Die Enum `Examtype` enthält nur die beiden Werte `Oral` und `Written` in dieser Reihenfolge.

WrittenExam

Die Klasse `WrittenExam` implementiert das Interface `Exam`. Implementieren Sie die dazu notwendigen Getter und die dazugehörigen Objektvariablen. In `getExamRooms` und `getStudents` erzeugen Sie jeweils ein neues `HashSet` aus dem Wert der jeweiligen Objektvariable und geben dieses zurück. Andernfalls könnte ein Aufrufer die in der Objektvariable gespeicherte Menge verändern (Stichwort: Referenzsemantik).

Die Methode `validate` wird weiter unten beschrieben.

Außerdem gibt es noch die Objektvariable

```
private LocalDateTime startDateTime;
```

mit einem Getter und einem Setter. Der einzige Konstruktor ist:

```
public WrittenExam(final int duration, final Lecturer examiner);
```

Denken Sie daran im Konstruktor oder gleich bei der Definition die beiden Mengen mit einem leeren `HashSet` zu initialisieren. Studierende können Sie dann mit den beiden Methoden

```
public void addStudent(final Student student);
public void addStudents(final Set<Student> students);
```

hinzufügen. Nutzen Sie im Rumpf einfach die passende `Set`-Methode.

Um einen Raum hinzuzufügen, implementieren Sie die Methode

```
public void addRoom(final Room room);
```

Eine Besonderheit dieser Methode ist, dass obwohl der Parameter den Typ `Room` hat, nur ein `ExamRoom` hinzugefügt werden darf. Dies können Sie überprüfen mit dem Operator `instanceof`. Ist der Parameter `room` ein `ExamRoom`, dann hat das Ergebnis von `room instanceof ExamRoom` den Wert `true`, anderenfalls `false`. Dann soll einfach nichts gemacht werden.

Eine weitere Besonderheit ist, dass diese Methode einen bereits enthaltenen Raum ersetzen soll. Mit der `Set`-Methode `add` geht das übrigens nicht automatisch.

Mit der Methode

```
public void removeRoom(final Room room);
```

kann ein Raum wieder entfernt werden.

Die Methode `validate` aus dem Interface implementieren Sie so, dass Sie `true` genau dann zurück geben, wenn es einen Prüfer und eine festgelgte Zeit gibt (`!= null`) und die Summe der Sitzplätze für Prüfungen (`seatsForExam`) mindestens so groß ist wie die Anzahl der für die Prüfung angemeldeten Studierenden.

`OralExam`

Die Klasse `OralExam` implementiert das Interface `Exam`. Nachdem bei mündlichen Prüfungen jeder Studierende eine andere Startzeit hat und auch in einem anderen Raum sitzen könnte, soll diese Information in den zwei `Maps`

```
private final Map<Student, LocalDateTime> startDateTimes;
private final Map<LocalDateTime, Room> rooms;
```

verwaltet werden. Für jeden Studierenden wird, sobald er sich für die Prüfung anmeldet, ein Eintrag in `startDateTimes` mit dem Datum und der Uhrzeit erzeugt. An anderer Stelle werden die Studierenden nicht mehr gespeichert. Für jede Startzeit wird, eventuell erst später, ein Eintrag in `rooms` erzeugt in dem der Raum eingetragen wird, in dem genau diese Prüfung stattfindet. Denken Sie daran die beiden Objektvariablen gleich sinnvoll zu initialisieren.

Der Konstruktor

```
public OralExam(final int duration, final Lecturer examiner);
```

initialisiert die weiteren Objektvariablen. Die Methode

```
public LocalDateTime getStartDateTime(final Student student);
```

gibt das Datum und die Uhrzeit der Prüfung des übergebenen Studierenden zurück. Falls es keinen Eintrag gibt, ist das Ergebnis `null`. Die Methode

```
public Room getExamRoom(final Student student);
```

macht das analog für den Raum. Mit den Methoden

```
public void addStudent(final LocalDateTime localDateTime, final Student student);
public void addRoom(final LocalDateTime localDateTime, final Room room);
```

wird ein Studierender bzw. ein Raum hinzugefügt. Mit der Methode `addStudent` kann die Prüfungszeit eines Studierenden auch verschoben werden! Überlegen Sie sich dazu wie das sinnvoll mit den vorhandenen `Map`-Methoden gemacht werden kann. Mit der Methode `addRoom` kann ein anderer Raum in einem vorhandenen Eintrag ersetzt werden. Überlegen Sie auch hier, welche Methode das macht (Ein Blick in die Dokumentation verschafft oft Klarheit ;-)).

Die Methode

```
public Room removeRoom(final LocalDateTime localDateTime);
```

entfernt einen Eintrag aus `rooms`.

Die Methode `validate` soll überprüfen ob es einen Prüfer gibt und ob die beiden `Maps` `startDateTime` und `rooms` "zusammenpassen", d.h. ob für jede Uhrzeit die in `startDateTime` auftaucht ein Eintrag in `rooms` vorhanden ist und umgekehrt. Nur dann ist das Ergebnis `true`.

Lecture

Die Klasse `Lecture` bringt nun alles in einer Lehrveranstaltung zusammen. Dazu hat sie folgende private und nach Möglichkeit unveränderbare Objektvariablen:

```
String lectureName;
Lecturer lecturer;
Room room; // der Vorlesungsraum
Set<Group> groups;
Examtype examtype;
Set<Student> students;
Exam exam;
```

Bis auf `students` und `exam` sollen alle Objektvariablen durch Konstruktorparameter mit den Werten versorgt werden. Das `exam` soll zu Beginn den Wert `null` bekommen und `students` soll ein leeres `HashSet` sein.

Für alle Objektvariablen, außer `examtype` und `exam`, soll es einen Getter geben. Denken Sie bei den `Sets` wieder daran, eine Kopie des `Sets` zurück zu geben.

Zum Hinzufügen von Studierenden gibt es die Methode

```
public void addStudents(final Set<Student> students);
```

Diese fügt zu den bereits schon gemeldeten Studierenden die übergebenen dazu. Für `exam` soll es einen Setter geben:

```
public boolean setExam(final Exam exam);
```

Dieser muss sicherstellen, dass der `examtype` (mündlich oder schriftlich) zum übergebenen Parameter passt (verwenden Sie dazu unter anderem `instanceof`). Passt es nicht zusammen, soll die Prüfung nicht gesetzt werden und das Ergebnis ist `false`. Wenn alles passt, ist das Ergebnis `true` und die Prüfung wurde gesetzt. Statt einem Getter gibt es die Methode

```
public Exam resetExam();
```

die die Objektvariable `exam` wieder auf `null` setzt und den bisherigen Wert zurück gibt. Die Idee ist dabei, dass Sie das `exam` zunächst von der Lehrveranstaltung entfernen, es dann bearbeiten (z.B. Räume hinzufügen) und anschließend die überarbeitete Prüfung wieder neu mit dem Setter zuweisen.

Für einen Studierenden kann mit den Methoden

```
public LocalDateTime getStartDateTime(final Student student);
public Set<Room> getExamRooms(final Student student);
```

die Prüfungszeit und der Raum berechnet werden. In beiden Methoden müssen Sie unterscheiden ob es eine mündliche oder schriftliche Prüfung ist, da die beiden Klassen ja unterschiedliche Methoden für Zeit und Räume zur Verfügung stellen. Nachdem für eine schriftliche Prüfung mehrere Räume Ergebnis sein können, wird der eine Raum bei einer mündlichen Prüfung auch, als einziger Wert, in eine Menge gepackt. Sonst müsste es verschiedene Methoden für die Räume in der Klasse `Lecture` geben.

Schließlich gibt es auch in `Lecture` eine Methode

```
public boolean validate();
```

Diese überprüft ob die gemeldeten Studierenden in den Raum passen und validiert die Prüfung, sofern vorhanden, mit deren `validate`-Methode. Ist keine Prüfung vorhanden, ist das auch ok. Nur wenn alles passt, ist das Ergebnis `true`. Ein Zusammenhang zwischen den Studierenden, die für die Lehrveranstaltung angemeldet sind und denen, die für die Prüfung angemeldet sind, wird nicht überprüft.

`Summerterm2018`

Die Klasse `Summerterm2018` enthält die beiden öffentlichen **Klassen**variablen `seiiib` und `algdatii`, die zwei Lehrveranstaltungen repräsentieren. Initilialisieren Sie die beiden Variablen in einem **statischen Initializer**, so dass die beiden folgendes repräsentieren:

`seiiib`

- Name: Softwareentwicklung II (IB)
- Dozent: Prof. Dr. Oliver Braun
- Raum: R1.006 mit 60 Plätzen
- angeboten für IB
- mit schriftlicher Prüfung
- Teilnehmer and der LV: Helga Meier und Helge Mayer
- Schriftliche Prüfung
  - Prüfer: Prof. Dr. Oliver Braun
  - Prüfungsraum: R1.046, 120 Sitzplätze aber nur 57 für Prüfungen
  - am 10.07.2018 um 10:30
  - Teilnehmer an der Prüfung: Helga Meier und Helge Mayer

`algdatii`

- Name: Algorithmen und Datenstrukturen II
- Dozent: Prof. Dr. Oliver Braun
- Raum: R0.010 mit 40 Plätzen
- angeboten für IC und IF
- mit mündlicher Prüfung
- Teilnehmer and der LV: Christa Schmidt und Christoph Schmitt
- Mündliche Prüfung
  - Prüfer: Prof. Dr. Oliver Braun
  - Teilnehmer an der Prüfung: Christa Schmidt und Christoph Schmitt
  - Zeiten:
    * Christa Schmidt: 03.07.2018, 08:00
    * Christoph Schmitt: 03.07.2018, 08:20
  - beide im Raum R3.013, 10 Sitzplätze, 5 für Prüfungen

Wenn Sie die Klasse `Summerterm2018` um eine `main`-Methode erweitern und im statischen Initializer am Beginn einen Breakpoint setzen, können Sie sich im Debugger anschauen, wie die beiden `Lectures` schrittweise zusammengebaut werden.