4 Optionale Repetitionsaufgabe: Raumverwaltung – Teil 1 von 5

4.1 Lernziele

- Repetition bereits behandelter Themen zur Festigung und Vertiefung Ihrer Kenntnisse.

4.2 Grundlagen

Beginnend mit dieser Übungseinheit erhalten Sie jede Woche eine optionale Zusatzaufgaben zur Repetition aller bisher behandelter Themen. Diese Zusatzaufgaben sind unabhängig von den restlichen Aufgaben dieser Woche. Sie bauen aber ihrerseits aufeinander auf.

Optionale Aufgaben sind **nicht** Bestandteil der Besprechungen.

4.3 Aufgaben

a.) Erstellen Sie eine nicht spezialisierbare Klasse `Raum` mit je einem Attribut für die Raumnummer (Ganzzahl) und der maximalen Platzanzahl (Kapazität). Nur Klassen im selben Package `ch.hslu.oop.rv` sollen Raumobjekte erzeugen können. Die Raumnummer und die Platzanzahl sollen direkt über den Konstruktor einmalig (immutable) gesetzt werden. Beachten Sie die Datenkapselung.

b.) Testen Sie mit einem Unit-Test, ob der Konstruktor die Attribute korrekt setzt.

c.) Prüfen Sie im Konstruktor die Parameter, so dass nur Raumnummern im Bereich von `100` bis `999` und nur Kapazitäten grösser als zwei Plätze möglich sind. Werfen Sie im Fehlerfall eine sinnvolle Exception mit individueller Meldung. Testen Sie mit drei Unit-Tests alle (Grenz-)Fälle.

d.) Implementieren Sie den equals-Contract auf der Klasse `Raum`. Zwei Räume sollen gleich sein, wenn die Raumnummer identisch ist.

3 Optionale Repetitionsaufgabe: Raumverwaltung – Teil 2 von 5

3.1 Lernziele

- Repetition bereit behandelter Themen zur Festigung und Vertiefung.

3.2 Grundlagen

Diese Zusatzaufgaben sind unabhängig von den restlichen Aufgaben dieser Woche. Sie baut aber auf der optionalen Repetitionsaufgabe der letzten Woche auf.

Optionale Aufgaben sind **nicht** Bestandteil der Besprechungen.

3.3 Aufgaben

a.) Erstellen Sie eine Enumeration, mit welcher Stati für einen **freien**, einen **belegten** und einen **gesperrten** Raum dargestellt werden können.

b.) Ergänzen Sie auf der Klasse **`Raum`** ein Attribut für den Raumstatus mit der Enumeration von a). Nur Klassen im selben Package sollen das Attribut setzen, aber alle anderen sollen es lesen können. Neue erstellte Räume sollen immer im Status '**frei**' sein.

c.) Implementieren Sie eine Klasse **`RaumVerwaltung`** im selben Package wie die Klasse **`Raum`**, welche eine Datenstruktur verwendet, um eine Menge von **`Raum`**-Objekten zu speichern. Auf die Räume soll mittels der Raumnummer schnell zugegriffen werden können! Die Datenstruktur soll die Einträge implizit nach der Raumnummer sortieren.

d.) Erzeugen Sie für erste Tests im Konstruktor der Klasse **`RaumVerwaltung`** einen Raum mit Nummer **`603`** und einer Platzanzahl von **`12`**. Legen Sie diesen Raum in der Datenstruktur von c) ab.

e.) Ergänzen Sie auf der Klasse **`RaumVerwaltung`** eine Methode, welche anhand einer Raumnummer den entsprechenden Raum (als Objekt) zurückliefert. Überlegen Sie sich, was im Fehlerfall passieren soll, und denken Sie an die Datenkapselung. Testen Sie das gewünschte Verhalten Ihrer Klasse mit zwei Unit-Tests.

3 Optionale Repetitionsaufgabe: Raumverwaltung – Teil 3 von 5

3.1 Lernziele

- Repetition bereit behandelter Themen zur Festigung und Vertiefung.

3.2 Grundlagen

Diese Zusatzaufgaben sind unabhängig von den restlichen Aufgaben dieser Woche. Sie baut aber auf der optionalen Repetitionsaufgabe der letzten Woche auf.

Optionale Aufgaben sind **nicht** Bestandteil der Besprechungen.

3.3 Aufgaben

a.) Testen Sie die `equals()`-Implementation auf der Klasse `Raum` mit **EqualsVerifier**.

b.) Fügen Sie der Klasse `Raum` eine öffentliche Methode hinzu, welche als `boolean` zurückliefert, ob ein Raum frei ist. Testen Sie diese Methode mit Unit-Tests.

c.) Implementieren Sie auf der Klasse `RaumVerwaltung` eine Methode, über welche für eine bestimmte Anzahl Personen automatisch ein passender Raum ausgewählt und reserviert wird. Als Rückgabetyp verwenden Sie die Klasse `Raum`. In einem ersten Schritt darf (zur Vereinfachung!) der erste genügend grosse (und natürlich freie) Raum gewählt werden.

d.) Ergänzen Sie auf der Klasse `RaumVerwaltung` eine Methode, mit welcher ein reservierter Raum wieder freigegeben wird. Wählen Sie einen geeigneten Rückgabetyp, um den Erfolg der Operation mitteilen zu können.

3 Optionale Repetitionsaufgabe: Raumverwaltung – Teil 4 von 5

3.1 Lernziele

- Repetition bereit behandelter Themen zur Festigung und Vertiefung.

3.2 Grundlagen

Diese Zusatzaufgaben sind unabhängig von den restlichen Aufgaben dieser Woche. Sie baut aber auf der optionalen Repetitionsaufgabe der letzten Woche auf.

Optionale Aufgaben sind **nicht** Bestandteil der Besprechungen.

3.3 Aufgaben

a.) Ergänzen Sie im Konstruktor der Klasse **`RaumVerwaltung`** die folgenden Räume als Testdaten:

Raumnummer	Kapazität	Hinweis
600	18	
602	6	
603	12	Schon vorhanden aus einer früheren Aufgabe
605	24	
610	12	

b.) Erstellen Sie eine Klasse **`Demo`** mit einer **`main(…)`**-Methode. Instanzieren Sie in dieser ein Objekt **`RaumVerwaltung`** und reservieren Sie je einen Raum für **`11`**, **`6`** und **`17`** Personen. Geben Sie danach ein Liste aller Räume auf der Konsole aus, und prüfen Sie, ob nun drei Räume belegt sind.

c.) Optimieren Sie die Implementation der Methoden auf der Klasse **`RaumVerwaltung`**. Überarbeiten Sie die Reservation derart, dass für eine Anforderung der jeweils kleinste noch passende Raum gewählt wird.

d.) Überladen Sie auf der Klasse **`Raum`** (und ggf. weiteren Klassen) die notwendigen Methoden, damit ein **`Raum`** einfach mit **`System.out.println(raum)`** in einem für Logging geeigneten Format ausgegeben werden kann.

e.) Überladen Sie auf der Klasse **`RaumVerwaltung`** die Methode zur Freigabe einer Reservation so, dass damit ein Raum auch nur anhand seiner Raumnummer freigegeben werden kann.

3 Optionale Repetitionsaufgabe: Raumverwaltung – Teil 5 von 5

3.1 Lernziele

- Repetition bereits behandelter Themen zur Festigung und Vertiefung.

3.2 Grundlagen

Diese Zusatzaufgaben sind unabhängig von den restlichen Aufgaben dieser Woche. Sie baut aber auf der optionalen Repetitionsaufgabe der letzten Woche auf.

Optionale Aufgaben sind **nicht** Bestandteil der Besprechungen.

3.3 Aufgaben

a.) Erstellen Sie eine spezialisierte Event-Klasse, welche als zusätzliche Attribute einen Raum und eine Anzahl Plätze enthalten kann.

b.) Erstellen Sie ein Functional Interface für den Event-Listener von a), und implementieren Sie alles Notwendige, so dass die Klasse `RaumVerwaltung` zu einer Event-Quelle wird.

c.) Versenden Sie bei einer Reservation und bei einer Freigabe einen Event. Registrieren Sie in der `Demo`-Klasse mit möglichst minimalem Aufwand einen Listener, welcher die gefeuerten Events in nachvollziehbarer Form per SLF4J/LogBack ausgibt (mit `INFO`-Level, wird auf der Konsole sichtbar sein).

d.) Implementieren Sie einen JUnit-Testfall, welcher automatisch prüft, ob bei einer Reservation ein entsprechender Event ausgelöst wird.