Funktionale und objektorientierte Programmierkonzepte Übungsblatt 08

TECHNISCHE UNIVERSITÄT DARMSTADT

Rückfragen zu diesem Übungsblatt vorzugsweise im moodle-Forum zu diesem Blatt!

Wintersemester 21/22	v1.0.2
Themen:	`assert` und Exceptions
Relevante Foliensätze:	05
Abgabe der Hausübung:	14.01.2022 bis 23:50 Uhr

H Hausübung 8 — Gesamt 28 Punkte

Exceptions

Wir verfolgen „Abschreiben“ und andere Arten von Täuschungsversuchen. Disziplinarische Maßnahmen treffen nicht nur die, die abschreiben, sondern auch die, die abschreiben lassen. Allerdings werden wir nicht unbedingt zeitnah prüfen, das heißt, es hat noch nichts zu bedeuten, wenn Sie erst einmal nichts von uns hören.

Verbindliche Anforderungen für die gesamte Hausübung: Auch in dieser Hausübung fordern wir wieder Dokumentation mittels `JavaDoc`. Informationen dazu finden Sie unter anderem auf Übungsblatt 03.
Für diese Hausübung müssen Sie alle Dateien wieder selber erstellen.

- Wenn die Rede davon ist, dass eine Klasse oder ein Interface `X` erstellt werden muss, muss diese/dieses in Datei `X.java` erstellt werden. Alle Dateien werden im Package `h08` im Verzeichnis `src/main/java` erstellt.
- Achten Sie darauf, alle Dateien und Identifier **exakt** wie angegeben zu benennen – das heißt: Verändern Sie nicht die Schreibweise – auch nicht die Groß- und Kleinschreibung, sofern dies nicht explizit gefordert wird. Beachten Sie unsere Informationen in Moodle zu unserem Vorgehen bei inkorrekten Schreibweisen.

H1: Vorbereitung — 2 Punkte

Schreiben Sie eine `public`-Klasse `TimeStamp` mit einer `private`-Objektvariable `lastUpdate` vom Typ `java.util.Calendar`.

Mit dem *Zeitpunkt eines `Calendar`-Objektes* meinen wir im Folgenden den Zeitpunkt, der durch dieses `Calendar`-Objekt repräsentiert wird, und mit *Zeitpunkt eines `TimeStamp`-Objektes* meinen wir den Zeitpunkt desjenigen `Calendar`-Objektes, auf das das Attribut `lastUpdate` in diesem `TimeStamp`-Objekt verweist. (Es wird sich im Weiteren herausstellen, dass `lastUpdate` niemals `null` sein kann, so dass der Zeitpunkt eines `TimeStamp`-Objektes immer wohldefiniert ist.)

Die `public`-Objektmethode `update` von `TimeStamp` hat keine Parameter und keine Rückgabe. Sie richtet mit dem parameterlosen `public`-Konstruktor von `java.util.GregorianCalendar` ein gregorianisches Kalenderobjekt ein (dieser Konstruktor hält den Zeitpunkt seines Aufrufs im Kalenderobjekt fest) und lässt `lastUpdate` darauf verweisen.

Der `public`-Konstruktor hat keine Parameter und ruft einfach nur `update` auf.

Die parameterlose `public`-Objektmethode `getTimeStamp` hat Rückgabetyp `Calendar` und liefert den momentan in diesem `TimeStamp`-Objekt gespeicherten Kalender zurück.

> ***Verständnisfrage am Rande* (0 Punkte):** Was denken Sie, warum wird in der Java-Standardbibliothek ein Unterschied gemacht zwischen einer Klasse `Calendar` und einer davon abgeleiteten Klasse `GregorianCalendar`, und warum ist `Calendar` abstrakt? (Schauen Sie sich auch unabhängig von dieser Verständnisfrage gerne einmal die Dokumentation von `java.util.Calendar` an.)

H2: Mit assert arbeiten — 3 Punkte

Fügen Sie in Klasse `TimeStamp` aus 1 eine zweite `public`-Objektmethode `update` ein, die einen Parameter vom formalen Typ `java.util.Calendar` und – wie die Methode `update` in 1 – keine Rückgabe hat.

Diese zweite Methode `update` überschreibt `lastUpdate` mit seinem aktualen Parameter, aber nur, wenn der aktuale Parameter einen Zeitpunkt bezeichnet, der weder früher als der momentane Zeitpunkt dieses `TimeStamp`-Objektes noch in der Zukunft liegt (also später als der Zeitpunkt, zu dem die Methode aufgerufen wird). Dazu fügen Sie als erste Anweisung in dieser zweiten Methode `update` eine `assert`-Anweisung ein, die genau das abprüft.

> ***Test zur eigenen Kontrolle* (0 Punkte):** Schreiben Sie in einer separaten Datei einen kleinen Test, in dem Sie die zweite Methode `update` zweimal aufrufen: zuerst mit einem Zeitpunkt, der durch `assert` gemäß obiger Vorgabe akzeptiert werden soll, dann mit einem Zeitpunkt, der durch `assert` gemäß obiger Vorgabe *nicht* akzeptiert werden soll. Um sicherzugehen, dass der logische Ausdruck in Ihrer `assert`-Anweisung vollständig korrekt ist, sollten Sie den nicht zu akzeptierenden Zeitpunkt im Quelltext variieren: einmal kompilieren und laufen lassen mit einem Zeitpunkt vor dem Zeitpunkt des `TimeStamp`-Objektes, dann einmal mit einem Zeitpunkt in der Zukunft.

***Unverbindliche Hinweise*:**

- Die beiden booleschen Objektmethoden `after` und `before` von `Calendar` haben jeweils ein `Calendar`-Objekt als Parameter und liefern genau dann `true` zurück, wenn der Zeitpunkt des `Calendar`-Objektes, mit dem die Methode aufgerufen wird, nach (bei `after`) bzw. vor (bei `before`) dem Zeitpunkt des aktualen Parameters lliegt.
- Die parameterlose Klassenmethode `getInstance` von Klasse `Calendar` liefert (einen Verweis auf) ein `Calendar`-Objekt zurück, dessen Zeitpunkt der Zeitpunkt des Aufrufs von `getInstance` ist.

> ***Verständnisfrage am Rande* (0 Punkte):** Der formale Typ des Parameters der Methoden `after` und `before` von `Calendar` ist nicht `Calendar`, wie man vielleicht vermuten würde, sondern `java.lang.Object`. Haben Sie eine Idee, warum das so ist?

H3: Exception-Klassen definieren — 3 Punkte

Schreiben Sie drei `public`-Klassen:

1. `BadUpdateTimeException` ist direkt von `java.lang.Exception` abgeleitet. Der `public`-Konstruktor hat zwei Parameter: einen vom Typ `Calendar` und einen vom Typ `boolean` (in genau dieser Reihenfolge). Dieser Konstruktor ruft denjenigen Konstruktor von `Exception` auf, der einen `String`-Parameter und sonst keine Parameter hat. Dieser Konstruktor setzt mit seinem aktualen Parameterwert die Botschaft (*message*) der Exception.

   Falls der boolesche aktuale Parameterwert `true` ist, soll die Botschaft mit „Update time is earlier than the last update: “ beginnen, ansonsten mit „Update time is in the future: “ (jeweils ein Leerzeichen nach dem Doppelpunkt!). Zur Vervollständigung der Botschaft wird der Zeitpunkt des `Calendar`-Objektes im Format „dd.mm.yyyy / hh:mm:ss:mmm!“ angehängt, wobei jeder Buchstabe für eine Ziffer (außer es handelt sich um eine einstellige bzw. bei Millisekunden auch zweistellige Zahl, dann soll auf die führende(n) Null(en) verzichtet werden, z.B. September muss als ”9”, nicht als ”09”dargestellt werden), das erste „mm“ für den Monat, das zweite für die Minute und das „mmm“ für die Millisekunde steht. Für die Stunde „hh“ soll das 24h-Format gewählt werden.
2. `UpdateTimeBeforeLastUpdateException` ist direkt von `BadUpdateTimeException` abgeleitet. Der `public`-Konstruktor von `UpdateTimeBeforeLastUpdateException` hat einen Parameter vom formalen Typ `Calendar` und ruft den Konstruktor von `BadUpdateTimeException` mit dem eigenen aktualen Parameter und `true` auf.
3. `UpdateTimeInTheFutureException` ist völlig analog zu `UpdateTimeBeforeLastUpdateException`, nur dass `true` durch `false` ersetzt ist.

H4: Exception werfen — 4 Punkte

Fügen Sie in Klasse `TimeStamp` fünf rückgabelose `public`-Methoden `updateWithExc1`...`updateWithExc5` ein, die völlig analog zur Methode `update` aus 2 definiert sind und das gleiche tun, außer dass sie jeweils eine Exception werfen in der Situation, in der die `assert`-Anweisung in `update` aus 2 den Prozess beendet. In den Methoden im Folgenden, in denen `UpdateTimeInTheFutureException` bzw. `UpdateTimeBeforeLastUpdateException` geworfen werden sollen (also in den Methoden 1-3), sollen sie natürlich genau in den Situationen geworfen werden, die der jeweilige Name nahelegt. Im Einzelnen:

1. In der `throws`-Klausel von `updateWithExc1` werden `UpdateTimeBeforeLastUpdateException` und `UpdateTimeInTheFutureException` deklariert und geworfen.
2. In der `throws`-Klausel von `updateWithExc2` wird `BadUpdateTimeException` deklariert, aber im Rumpf von `updateWithExc2` werden `UpdateTimeBeforeLastUpdateException` und `UpdateTimeInTheFutureException` geworfen.
3. In der `throws`-Klausel von `updateWithExc3` wird `Exception` deklariert, aber im Rumpf von `updateWithExc3` werden `UpdateTimeBeforeLastUpdateException` und `UpdateTimeInTheFutureException` geworfen.
4. In der `throws`-Klausel von `updateWithExc4` wird `Exception` deklariert, aber im Rumpf von `updateWithExc4` wird `BadUpdateTimeException` geworfen. Beim Werfen der `BadUpdateTimeException` soll hier die gleiche Fallunterscheidung wie in den Fällen 1-3 stattfinden, entsprechend muss der Konstruktor mit unterschiedlichem zweiten Parameter aufgerufen werden.
5. In der `throws`-Klausel von `updateWithExc5` wird `Exception` deklariert, und im Rumpf von `updateWithExc5` wird `Exception` geworfen. Hier muss die Botschaft ohne die Mithilfe des Konstruktors von `BadUpdateTimeException` erstellt werden.

***Verbindliche Anforderung*:** Keine der fünf Methoden `updateWithExc1`...`updateWithExc5` ruft eine andere dieser fünf Methoden direkt oder indirekt auf, das heißt, die Implementationen aller fünf Methoden sind unabhängig voneinander.

***Verständnisfrage am Rande* (0 Punkte):** Ginge es auch hier wie in 2, diese fünf Methoden oder zumindest eine davon einfach `update` zu nennen? Probieren Sie es aus! Schauen Sie sich dazu auch gerne nochmals Folien 83 ff. in Kapitel 03c an.

H5: Exception fangen — 6 Punkte

Schreiben Sie eine `public`-Klasse `TestTimeStampExceptions` mit fünf Methoden, `testCatch1`...`testCatch5`, die jeweils einen Parameter vom formalen Typ `TimeStamp`, einen Parameter vom formalen Typ `Calendar` und einen Parameter vom formalen Typ `int` haben (in genau dieser Reihenfolge).

Für jede dieser fünf Methoden gilt: Falls der aktuale Wert n des dritten Parameters einer der Werte $1 \ldots 5$ ist, ruft jede der Methoden `testCatch1`...`testCatch5` die Methode `updateWithExc`n einmal auf, fängt eventuelle Exceptions wie im Folgenden beschrieben und macht sonst nichts (für jeden anderen Wert von n macht die Methode gar nichts).

Genauer gesagt, steht in jeder dieser fünf Testmethoden `testCatch1`...`testCatch5` ein einzelner `try`-Block, und der besteht jeweils aus einer `switch`-Anweisung mit den fünf Optionen $1 \ldots 5$ (siehe Folien 214 ff. in Kapitel 03c). Option n beinhaltet den Aufruf von Methode `updateWithExc`n des ersten mit dem zweiten aktualen Parameter der jeweiligen Testmethode `testCatch1`...`testCatch5`. (Sie dürfen den – für alle fünf Testmethoden ja identischen – Inhalt des `try`-Blocks gerne in eine separate `private`-Methode auslagern.)

Die `catch`-Blöcke sind bei den einzelnen Testmethoden `testCatch1`...`testCatch5` unterschiedlich gewählt, machen aber alle ungefähr dasselbe. Jeder `catch`-Block zum oben erläuterten `try`-Block gibt eine Zeile auf der Konsole aus (mit Zeilenabschluss, also „`println`“). Der Beginn der Zeile ist die gewählte Option $1 \ldots 5$, gefolgt von einem Doppelpunkt (mit jeweils einem Leerzeichen vor und nach dem Doppelpunkt), dann der statische Typ der Exception, wie er im Kopf des `catch`-Blocks angegeben ist. Nach einem weiteren Doppelpunkt (ebenfalls mit jeweils einem Leerzeichen vor und nach dem Doppelpunkt) folgt der dynamische Typ der in diesem `catch`-Block gefangenen Exception. Nach einem Leerzeichen folgt dann noch die Message der Exception. Eine mögliche Ausgabe wäre z.B.:

“1 : UpdateTimeBeforeLastUpdateException : UpdateTimeBeforeLastUpdateException Update time is earlier than the last update: 1.1.2000 / 12:25:13:254!\n“

In der folgenden Auflistung ist jeweils die explizit im Kopf des `catch`-Blocks benannte Exception-Klasse gemeint (Hinweis: falls notwendig sollen Sie weitere catch-Blöcke hinzufügen, die nicht explizit in dieser Auflistung benannt sind. In diesen Fällen müssen Sie nichts ausgeben):

1. Im `try-catch`-Block von `testCatch1` gibt es je einen `catch`-Block für `UpdateTimeBeforeLastUpdateException` und `UpdateTimeInTheFutureException`.
2. Im `try-catch`-Block von `testCatch2` gibt es einen einzigen, gemeinsamen `catch`-Block für `UpdateTimeBeforeLastUpdateException` und `UpdateTimeInTheFutureException`, also beide im Kopf desselben `catch`-Blocks nacheinander genannt. Als statischer Typ soll hier in der Ausgabe “UpdateTimeBeforeLastUpdateException oder UpdateTimeInTheFutureException“ genommen werden.
3. Im `try-catch`-Block von `testCatch3` gibt es je einen `catch`-Block für `UpdateTimeBeforeLastUpdateException` und `BadUpdateTimeException` (warum unbedingt in dieser Reihenfolge?).
4. Im `try-catch`-Block von `testCatch4` gibt es einen `catch`-Block für `BadUpdateTimeException`.
5. Im `try-catch`-Block von `testCatch5` gibt es einen `catch`-Block für `Exception`.

> ***Unverbindlicher Hinweis*:** Bei einigen dieser Methoden `test1`...`test5` können Sie Downcast gemäß Kapitel 03b, Folien 169 ff. für die Identifizireung des dynamischen Typs verwenden.

> ***Tests zur eigenen Kontrolle* (0 Punkte):** Rufen Sie jede der Methoden `testCatch1`...`testCatch5` jeweils fünfzehnmal auf: dreimal mit jeder Option $1 \ldots 5$, und zwar jeweils einmal mit einem akzeptablen Zeitpunkt, einmal mit einem zu frühen Zeitpunkt und einmal mit einem zu späten Zeitpunkt.

> ***Verständnisfrage am Rande* (0 Punkte):**
> Die Zahlen `1`...`5` in den Methodennamen `updateWithExc1`...`updateWithExc5` kann man auf mindestens zwei verschiedene Weisen mit `updateWithExc` semantisch in Beziehung sehen. Welche sind das und welche davon war hier offenbar bei der Namensgebung `updateWithExc1`...`updateWithExc5` gemeint? Könnte so etwas in ähnlichen Situationen zu Missverständnissen führen?

H6: Exception weiterreichen — 2 Punkte

Fügen Sie in Klasse `TestTimeStampExceptions` zwei rückgabelose `public`-Methoden ein, die jeweils einen Parameter vom formalen Typ `TimeStamp` und einem Parameter vom formalen Typ `Calendar` in genau dieser Reihenfolge haben:

1. `testPass`: Die `throws`-Klausel deklariert `BadUpdateTimeException`. Die Methode ruft `updateWithExc1` des ersten mit dem zweiten eigenen aktualen Parameterwert auf, aber ***nicht*** in einem `try-catch`-Block, sondern ganz normal, als würde `updateWithExc1` keine Exception werfen (sollte durch den Compiler gehen!).
2. `testCatchPassed` hat keine `throws`-Klausel. Die Methode ruft `testPass` mit ihren eigenen beiden aktualen Parameterwerten auf, und zwar in einem `try-catch`-Block mit einem `catch`-Block, der `BadUpdateTimeException` fängt und (bis auf die Zahl und den Doppelpunkt mit dem Leerzeichen davor und danach am Beginn der Zeile) eine Zeile nach demselben Format wie in 5 ausgibt.

> ***Tests zur eigenen Kontrolle* (0 Punkte):** Erweitern Sie Ihre Tests zur eigenen Kontrolle in 5 um diese beiden Methoden.

H7: Raum-Management für die Themen-Sprechstunde — 8 Punkte

In dieser Aufgabe implementieren Sie eine Methode, mit welcher überprüft werden kann, ob der für die Themen-Sprechstunde genutzte Raum alle zur Themen-Sprechstunde angemeldeten Studierenden unter Einhaltung folgender Hygienerichtlinen aufnehmen kann:

- Jeder Raum darf zu maximal 50 Prozent besetzt werden. Das bedeutet, dass nur die Hälfte (abgerundet) der Sitzplätze zur Verfügung stehen.
- Alle Studierenden müssen einen Nachweis über eine Schutzimpfung, einen negativen Test oder eine Genesung (3G) vorlegen können.

In der Vorlage sind im Package `h08.roommanagement` bereits die Klassen `Room` und `Student` vorgegeben und müssen nicht von Ihnen implementiert werden:

Ein Objekt der `public`-Klasse `Room` stellt einen Raum dar. Die Klasse besitzt zwei `public-final`-Objektattribute: Das Objektattribut `name` ist vom statischen Typ `String` und enthält die Bezeichnung des Raumes – zum Beispiel `"S105/122"`. Das Objektattribut `numberOfSeats` ist vom Typ `int` und gibt die Anzahl der im Raum existierenden Plätze an.

Studierende werden mittels Objekten der `public`-Klasse `Student` dargestellt. Die Klasse beitzt ebenfalls zwei `public-final`-Objektattribute: Das Objektattribut `name` ist vom Typ `String` und enthält den Namen der Person – zum Beispiel `"Algo-Rith Mik"`. Das Objektattribut `hasCertificate` ist vom Typ `boolean`. `hasCertificate` ist genau dann `true`, wenn die Person einen Nachweis über eine Schutzimpfung, einen negativen Test oder eine Genesung vorlegen kann.

Beide Klassen haben jeweils einen `public`-Konstruktor, der alle Objektattribute direkt setzt.

H7.1: Exception-Klassen — 4 Punkte

Die folgenden Klassen sind in Package `h08.roommanagement` zu platzieren.

Um anzuzeigen, dass ein Raum nicht ausreichend viele Sitplätze hat, soll eine direkt von `RuntimeException` abgeleitete `public`-Klasse namens `InsufficientNumberOfSeatsException` erstellt werden, dessen einziger `public`-Konstruktor zwei Parameter hat: Der erste Parameter `room` ist vom formalen Typ `Room` und referenziert das Objekt, das den Raum repräsentiert, der nicht ausreichend viele Sitzplätze besitzt. Der zweite Parameter namens `numberOfMissingSeats` ist vom formalen Typ `int`. `numberOfMissingSeats` ist gleich der Anzahl an Plätzen, die zusätzlich zur Verfügung stehen müssen, damit alle Studierenden im Raum aufgenommen werden können. Die Methode `getMessage` der Klasse `InsufficientNumberOfSeatsException` soll den String `"{room} has not enough seats"` zurückliefern. Der Substring `"{room}"` soll durch das Objektattribut `name` des ersten aktualen Parameters ersetzt werden. Außerdem implementieren Sie in Klasse `InsufficientNumberOfSeatsException` eine parameterlose `public`-Objektmethode namens `getNumberOfMissingSeats` mit Rückgabetyp `int`, die den Wert des zweiten aktualen Parameters des Konstruktors liefert.

Des Weiteren soll, um anzuzeigen, dass Studierende keinen Nachweis über eine Impfung, eine Genesung oder einen Test vorlegen können, eine direkt von `Exception` abgeleitete `public`-Klasse namens `NoCertificateException` erstellt werden. Der einzige `public`-Konstruktor hat einen Parameter vom formalen Typ „Array von `Student`“, welcher die Studierenden enthält, die keinen entsprechenden Nachweis vorlegen können. Die Methode `getMessage` der Klasse `NoCertificateException` soll den String `"{names} has/have no certificate(s)"` zurückliefern. Der Substring `"{names}"` soll durch die Konkatenation der Werte des Objekattributs `names` der Komponenten im Array ersetzt werden, wobei als *Separator* zwischen den einzelnen Substrings der Substring `", "` verwendet werden soll.

> ***Verbindliche Anforderung*:** Sowohl `InsufficientNumberOfSeatsException` als auch `NoCertificateException` darf die Methode `getMessage` aus Klasse `Exception` nicht überschreiben.

H7.2: Überprüfung — 4 Punkte

Schreiben Sie in Klasse `h08.Main` eine `public`-Klassenmethode `checkRegistration`, welche einen ersten Parameter vom formalen Typ „Array von `Student`“ und einen zweiten Parameter vom formalen Typ `Room` hat. Das Array referenziert alle Studierenden, die sich für die Themen-Sprechstunde angemeldet haben. Der gegebene Raum ist der Raum, in welchem die Themen-Sprechstunde stattfinden soll.

> ***Beispiel*:** Nach den Hygienerichtlinen dürfen bei der Themen-Sprechstunde in der Regel (sofern die Themen-Sprechstunde nicht ausnahmsweise in einem anderen Raum stattfindet) maximal $\lfloor 372 \cdot 0.5 \rfloor = 186$ Studierende in Präsenz teilnehmen. Bei 203 angemeldeten Studierenden müssten mindestens $203 \cdot 2 = 406$ Plätze vorhanden sein. Das sind $406 - 372 = 34$ Plätze, welche für die Themen-Sprechstunde fehlen würden.

Zuerst wird in Methode `checkRegistration` geprüft, ob der gegebene Raum mit Sitzplatzbeschränkung alle im gegebenen Array enthaltenen Studierenden aufnehmen kann. Es darf davon ausgegangen werden, dass die Anzahl der im Array enthaltenen Studierenden gleich der Länge des Arrays ist. Der Wert des Objektattributs `numberOfSeats` in Klasse `Room` ist gleich der Anzahl der Sitzplätze **vor** der Beschränkung auf 50 Prozent der Sitzplätze.

Im Fall, dass der gegebene Raum nicht alle Studierenden aufnehmen kann, soll eine Exception vom Typ `InsufficientNumberOfSeatsException` geworfen werden.

Das Objekt vom Typ `InsufficientNumberOfSeatsException` wird wie folgt konstruiert: Als erster Parameter wird der der Methode `checkRegistration` gegebene Raum übergeben. Als zweiter Parameter wird die Anzahl an Plätzen übergeben, die, um alle Studierenden aufzunehmen, zusätzlich zu den im Raum zur Verfügung stehenden Plätzen benötigt werden.

Andernfalls wird überprüft, ob jeder der gegebenen Studierenden einen Nachweis über eine Impfung, einen Test oder eine Genesung vorlegen kann. Wenn Studierende diese Bedingung nicht erfüllen, soll eine Exception vom Typ `NoCertificateException` geworfen werden, welche mit einem Array, das alle Studierenden, die diese Bedingung nicht erfüllen, konstruiert wird. Die Länge des Arrays darf nicht größer als die Anzahl der darin enthaltenen paarweise verschiedenen Elemente sein.

Wenn keine der beiden Exceptions geworfen wird, soll die Methode nichts weiter tun.

***Verständnisfragen am Rande* (0 Punkte):**

- Wenn Methode `checkRegistration` im Normalfall (keine Exception) nichts tut: Wieso kann es dennoch sinnvoll sein, eine solche Methode zu implementieren und aufzurufen? Tipp: Schauen Sie sich zum Beispiel die Klassenmethode `Objects.requireNonNull` an.

Unsere Exception-Klassen `InsufficientNumberOfSeatsException` und `NoCertificateException` sind direkt von zwei verschiedenen Klassen abgeleitet.

- Welche Exception-Klassen müssen in der `throws`-Klausel angegeben werden?
- Welche Exception-Klassen müssen mittels `try-catch` behandelt werden, wenn diese nicht gefangen oder weitergeleitet werden?
- Was passiert, wenn eine nicht notwendigerweise zu behandelende Exception nicht behandelt wird?

***Tests zur eigenen Kontrolle* (0 Punkte):** Rufen Sie die Methode `checkRegistration` mit verschiedenen aktualen Parametern auf, so dass

1. die Anmeldung akzeptiert wird (keine Exception),
2. die Anmeldung wegen zu hoher Nachfrage abgelehnt wird (`InsufficientNumberOfSeatsException`) und
3. die Anmeldung wegen eines nicht vorhandenen Zertifikats abgelehnt wird (`NoCertificateException`).

Prüfen Sie auch, ob die generierten Botschaften (Methode `getMessage`) der Exceptions sowie die Rückgabe von `getNumberOfMissingSeats` korrekt ist.