Übungsblatt 3 – Teil 1

Ausgabe: 10.05.2014
Abgabe: 21.05.2014

Aufgabe 1: Exceptions - Theorie — 15 Punkte

a) Nennen Sie 2 Vorteile, warum Exceptions statt Fehlercodes verwendet werden sollten.

b) Erklären Sie die catch-or-throw-Regel anhand eines einfachen selbst entworfenen Beispiels.

Aufgabe 2: Exceptions — 40 Punkte

Entwickeln Sie ein Programm, mit dem angehende Piloten einer Sightseeing Tour den Ablauf von Touristenflügen lernen. Die Piloten fliegen vom Flughafen aus eine Runde über Mannheim. Da der Flughafen zu allen Seiten zwei Kilometer zur Stadt entfernt ist, können die Piloten auf diesen zwei Kilometern so tief fliegen wie sie möchten, um das Flugzeug starten und wieder landen zu können. Während sie über die Stadt fliegen, muss allerdings eine Mindesthöhe eingehalten werden. Da es sich nur um kleine Touristenflugzeuge handelt, darf durchgehend eine Maximalhöhe nicht überschritten werden, um größere Flugzeuge nicht zu behindern. Des Weiteren muss darauf geachtet werden, dass das Flugzeug nur mit geschlossenen Türen los fliegen darf. Die Türen dürfen nur geöffnet werden, wenn das Flugzeug auf dem Boden ist und sich nicht mehr bewegt.

a) Schreiben Sie die Klasse `FlightRoute`, in der zu jeder Route eingetragen werden kann, wie viele Kilometer sie lang ist, was die Mindesthöhe über der Stadt ist und wie hoch das Flugzeug höchstens fliegen darf. Stellen Sie sicher, dass eine Flugroute valide Daten hat. Werden ungültige Daten übergeben, muss eine `SimulatorConfigurationException` geworfen werden, die den Vorgang abbricht. Diese Exception muss auch implementiert werden.

b) Schreiben Sie nun die Klasse des Flugzeugs. Für das Flugzeug liegt bereits folgendes Interface vor:

```
public interface Plane {

/**
* Öffnet die Türen des Flugzeugs. Damit die Türen geöffnet werden können,
* muss sich das Flugzeug auf dem Boden befinden und darf sich nicht
* mehr bewegen.
* @throws GeneralFlightSimulatorException
*          Wenn das Flugzeug noch in der Luft ist oder noch nicht
*          still steht.
```

```
*/
public void openDoors() throws GeneralFlightSimulatorException;

/**
* Schließt die Türen des Flugzeugs.
*/
public void closeDoors();

/**
* Hält das Flugzeug an, wenn es gerade auf dem Boden fährt.
* @throws GeneralFlightSimulatorException
*         Wenn das Flugzeug in der Luft ist
*/
public void stop() throws GeneralFlightSimulatorException;

/**
* Lässt das Flugzeug einen weiteren Kilometer fliegen, der Höhenunterschied
* wird als Parameter angegeben.
* @param  additionalHeight
*         Der Höhenunterschied, den das Flugzeug in diesem Kilometer höher /
*         niedriger fliegt als zuvor. Kann positiv oder negativ sein.
* @throws GeneralFlightSimulatorException
*         Falls beim Fliegen Probleme auftauchen.
*/
public void flyNextKilometer(int additionalHeight) throws
      GeneralFlightSimulatorException;
}
```

Bei jeder Methode muss überprüft werden, ob diese zum Zeitpunkt des Aufrufs zulässig ist. Eine zulässige Simulationsrunde beginnt mit dem Schließen der Türen. Anschließend darf das Flugzeug losfliegen. Ab dem Erreichen des zweiten Kilometer muss es die Mindesthöhe einhalten, bis es nur noch zwei Kilometer vom Flughafen entfernt ist. Ab diesem Zeitpunkt kann es wieder tiefer fliegen um schließlich zu landen. Bevor letztendlich die Türen wieder geöffnet werden können, muss das auf dem Boden rollende Flugzeug noch angehalten werden. Um den Passagieren einen ruhigen Flug zu gewährleisten darf sich pro Kilometer die Höhe maximal um 100m ändern. Implementieren Sie die `PlaneTooHeighException` und die `PlaneTooLowException`, die beide von `GeneralFlightSimulatorException` ableiten. Diese sollen geworfen werden, wenn das Flugzeug zu tief oder zu hoch fliegt und die aktuelle Flughöhe dabei ausgeben. Bei allen anderen ungültigen Methodenaufrufen, auch wenn das Flugzeug mehr Kilometer fliegt als in der Route vorgesehen, soll die `GeneralFlightException` geworfen werden.

Schreiben Sie ein geeignetes Programm, um die Simulation zu testen. Zuerst muss die Flugroute festgelegt werden, anschließend folgen Simulationsrunden bis das Programm beendet wird. Nach jeder Simulationsrunde wird vermerkt, ob diese erfolgreich beendet oder durch eine Execption unterbrochen wurde, anschließend wird eine neue Simulation gestartet. Wurde die Simulation, wenn Sie korrekt eingestellt war, durch eine Exception unterbrochen, wird vermerkt, welche Exception aufgetreten ist. Fangen Sie hierzu jede Exception einzeln ab.

Wird das Programm beendet, soll eine Bilanz gezeigt werden, wie oft die Simulation erfolgreich war und wie oft sie durch welche Exception unterbrochen wurde.

c) Schreiben Sie JUnit Tests für die Methode `flyNextKilometer`.