Aufgabenblatt 8

Einführung in die Programmierung 2

LVA-Nr. 185.A92
2018 S
TU Wien

Thema:

Code-Review, Statisches Programmverstehen

Ausgabe:

11. 6. 2018

Abgabe (Deadline):

18. 6. 2018, 6:00 Uhr

Lösungen hochladen und gelöste Aufgaben ankreuzen (TUWEL)

Skriptum:

Seiten 134 – 146
Aufgaben 4.17 – 4.35
Invarianten beziehen sich auf das Objekt bzw. seine Objektvariablen

Allgemeines zum Lösen von Aufgabenblättern

Verwenden Sie Ihr existierendes IntelliJ-IDEA-Projekt und lösen Sie darin die Aufgaben. Eine gute Lösung erfüllt die Vorgaben der Aufgabenbeschreibung, ist kurz und einfach gehalten, ist mit informativen Kommentaren versehen und wurde gut getestet. Objektvariablen sind `private`.

Das Projekt mit den gelösten Aufgaben muss rechtzeitig vor der Deadline als ZIP-Datei in TUWEL hochgeladen werden. Programmtext für das Testen soll im Projekt enthalten sein. Bei mehrfachem Hochladen zählt die zuletzt hochgeladene ZIP-Datei. Es gibt keine andere Möglichkeit zur Abgabe.

Jede gelöste Aufgabe muss in TUWEL angekreuzt werden. Lösungen angekreuzter Aufgaben müssen in der Übungseinheit präsentiert werden können. Nach der Deadline ist das Ändern der Kreuzchen nicht möglich.

Aufgabe 1 (verpflichtend, 20%, 1 Punkt) **unbedingt lösen**

Design-by-Contract: Schreiben Sie mindestens eine Vor- und eine Nachbedingung als Kommentar zur Methode `add` Ihrer Implementierung der Klasse `SongTree`. Schreiben Sie mindestens eine Invariante für die Klasse `SongTree`, oder die zugehörige Knotenklasse. Wenn Ihnen dabei die aktuelle Implementierung der Klasse bzw. Methode nicht mehr sinnvoll erscheint, ändern Sie sie. Sie können dabei Fehler in den Eingabe-Argumenten durch Werfen einer Exception melden.

Fragen

Wie leicht ist es für den Client, die Vorbedingung zu erfüllen? Sollte man stattdessen eine einfacher zu erfüllende Vorbedingung fordern, und dafür eine Überprüfung im Client einbauen? Falls Sie eine Exception geworfen haben: Wäre es sinnvoll gewesen, stattdessen von vornherein einen anderen Rückgabewert vorzusehen?

Aufgabe 2 (20%, 1 Punkt)

Schreiben Sie Zusicherungen (Vor- und Nachbedingungen wie in Hoare-Tripeln) an mehreren Stellen im Rumpf der Methode `add` Ihrer Implementierung der Klasse `SongTree` und der von ihr aufgerufenen Hilfsmethoden (auch in Hilfsklassen). Schreiben Sie Zusicherungen insbesondere unmittelbar vor und nach jeder Änderung im Kontrollfluß (z.b. bei einem `if` davor, am Anfang und Ende jedes Zweigs, und unmittelbar hinter der `if`-Anweisung), und leiten Sie so die Nachbedingung der Methode aus der Vorbedingung der Methode ab. Die Zusicherungen können in Form von Kommentaren oder `assert`-Anweisungen geschrieben werden, auch gemischt. Wenn Ihnen dabei die aktuelle Implementierung der Methode nicht mehr sinnvoll erscheint, ändern Sie sie.

Aufgabe 3 (20%, 1 Punkt)

Schreiben Sie eine Schleifeninvariante für eine Schleife der Methode `getLaenge` der Klasse `Playlist1` bzw. einer davon aufgerufenen Hilfsmethode (auch in einer Hilfsklasse). Die Schleifeninvariante soll dazu dienen, die Erfüllung der Nachbedingung dieser Methode zu beweisen; falls ihre aktuelle Implementierung dieser Methode rekursiv oder schlecht geeignet ist, implementieren Sie sie für dieses Beispiel neu, als Schleife. Schreiben Sie die Schleifeninvariante als `assert`-Anweisung; diese kann Hilfsmethoden aufrufen. Fügen Sie die Schleifeninvariante an den passenden Stellen in der Methode ein.

Aufgabe 4 (40%, 2 Punkte)

Auf TUWEL finden Sie eine Beispiellösung für die Teamaufgabe unter *Aufgabenblatt 8* als `Teamaufgabe (Musterloesung).zip`. Laden Sie sie herunter und überprüfen Sie `Junction.java` und `JunctionQuadTree.java` in einer Code-Review gemäß der unten stehenden Kriterien. Fügen Sie zu jedem Kriterium an den entsprechenden Stellen Kommentare in den Quellcode ein, in denen Sie erklären, wie und warum es an der Stelle nicht erfüllt ist; falls Sie zu einem Kriterium keine solchen Stellen finden, fügen Sie einen Kommentar ein, in dem Sie erklären, dass das Kriterium erfüllt ist. Ihre Code-Review-Kommentare sollten mit `// !! N)` beginnen, wobei `N` die Nummer des Kriteriums ist.

Dabei müssen Sie nicht beweisen, dass sie alle Stellen gefunden haben, an denen die Kriterien nicht erfüllt sind, sondern nur einen ehrlichen Versuch machen, alle solchen Stellen zu finden. Nach 90 Minuten Code-Review dürfen Sie aufhören, und dürfen trotzdem ankreuzen.

1. Werden nur nötige Eigenschaften vorausgesetzt (etwa kein Untertyp wo auch ein Obertyp reichen würde)?
2. Werden alle Daten validiert, die von außerhalb des Programms kommen könnten?

   `junctions.csv` ist "von außerhalb"
3. Sind Überprüfungen von Daten ausreichend restriktiv, sodass ein Eindringen von außen nicht möglich zu sein scheint?
4. Werden Überprüfungen von Daten nicht unnötig wiederholt und sind sie in sich konsistent?
5. Kommen keine unnötigen, möglicherweise nach Änderungen übriggebliebenen Programmteile vor, oder ähnlicher Code, der leicht zusammengefasst und vereinfacht werden könnte?
6. Ist die Sichtbarkeit von Programmteilen bestmöglich beschränkt?

Für die Abgabe laden Sie bitte 2 zip-Dateien hoch, eine zip-Datei für die Aufgaben 1,2,3 und eine zip-Datei für die Aufgabe 4. Der Name der zip-Datei für das Projekt mit den Aufgaben 1,2 und 3 ist *MatrNr* `-AB8.zip` (zB: `12345678-AB8.zip`). Der Name der zip-Datei für das Projekt mit der Aufgabe 4 ist *MatrNr* `-review.zip` (zB: `12345678-review.zip`).