Aufgabenblatt 4

Einführung in die Programmierung 2

LVA-Nr. 185.A92
2018 S
TU Wien

Thema:

mehrere Implementierungen eines Interfaces; dyn. Binden, toString, equals

Ausgabe:

30. 4. 2018

Abgabe (Deadline):

14. 5. 2018, 6:00 Uhr

Lösungen hochladen und gelöste Aufgaben ankreuzen (TUWEL)

Skriptum:

Seiten 73 – 83
Aufgaben 3.1 – 3.16

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, 40%, 2 Punkte) **unbedingt lösen**

Diese Aufgabe wird in der Ad-hoc-Aufgabe erweitert. Wenn Sie sie nicht machen, werden Sie die Ad-hoc-Aufgabe vermutlich nicht schaffen.

Schreiben Sie ein Interface `SongTreeNodable` mit den Methoden

```
SongTreeNodable add(Song song)
void print()
```

Implementieren Sie dieses Interface in zwei Klassen `SongTree1Binary` und `SongTree1Null`. Dabei repräsentiert `SongTree1Binary` einen nichtleeren Knoten eines binären Suchbaums und enthält einen `Song` und zwei (möglicherweise leere) Unterbäume, die `SongTreeNodable` implementieren. Und `SongTree1Null` repräsentiert einen leeren binären Suchbaum und hat keine Objektvariablen.

Ein Aufruf `mytree.add(mysong)` liefert einen binären Suchbaum, der genau die Songs aus `mytree` und zusätzlich noch `mysong` enthält. `mytree` selbst darf dabei verändert werden, muss es aber nicht (und wird es auch nicht in jedem Fall, insbesondere nicht, wenn `mytree` ein `SongTree1Null` ist).

siehe DynBindList, Vorlesung vom 23.4.2018

Implementieren Sie eine Klasse `SongTree1`, die sich für Aussenstehende wie `SongTree` aus Aufgabenblatt 3/2 verhält, intern aber `SongTree1Binary` und `SongTree1Null` verwendet.

Verwenden Sie in der Implementierung nicht `null` (sondern stattdessen ein Objekt der Klasse `SongTree1Null`), und verwenden Sie kein `if` oder `? :`, sondern stattdessen dynamisches Binden: Wenn sie z.B. bei einem leeren Teilbaum eine andere `print`-Aktion haben wollen als bei einem nicht-leeren Knoten, definieren Sie die `print`-Methoden für `SongTree1Null` entsprechend.

Unterschied bei `add`

Fragen (und Hinweise)

- Wozu ist der Rückgabewert des `add` in `SongTreeNodable` gut? Wäre es auch möglich, mit einem `add`, das den Unterbaum nicht zurückgibt, das Einfügen in den Baum zu implementieren, ohne auf ein `if` zurückgreifen zu müssen?

- Wo werden während dieser Aufgabe tatsächlich zur Laufzeit unterschiedliche Implementierungen derselben Methode von der selben Stelle aus aufgerufen?
- Was sind die Vor- und Nachteile der `SongTree1`-Implementierung im Vergleich zur Implementierung von `SongTree` oder der von `STree` im Skriptum?

Aufgabe 2 (20%, 1 Punkt)

Implementieren Sie die Methode `toString` in `SongTree1` und den für die Implementierung verwendeten Klassen. Verwenden Sie dabei dynamisches Binden (und rekursive Aufrufe) statt `if`, `? :`, oder `while`. Verwenden Sie `toString` für die Implementierung von `print` in `SongTree1`.

Fragen

- Muss man dafür `toString` zu `SongTreeNodable` hinzufügen? Darf man es? Begründen Sie Ihre Antwort.
- Vergleichen Sie `toString` von `SongTree1` mit der ursprünglichen Implementierung von `print`. Was sind die Unterschiede und Gemeinsamkeiten?

Aufgabe 3 (40%, 2 Punkte)

Implementieren Sie die Methoden `equals` und `hashCode` für `Song` und `SongTree1`, wobei zwei Songs als gleich gelten sollen, wenn sie den gleichen Titel, die gleiche Band und die gleiche Länge haben. Zwei `SongTree1`-Objekte sollen dann als gleich gelten, wenn sie die gleichen Songs enthalten, auch wenn die Baumstruktur unterschiedlich ist.

Fragen

- Unter der Annahme, dass die beteiligten Bäume halbwegs balanciert sind, und wir zwei gleiche, unterschiedlich strukturierte Objekte der Klasse `SongTree1` mit n Songs haben: Wie steigt der benötigte Extra-Speicher Ihrer `equals`-Implementation mit n an? Wie steigt die Laufzeit mit n an? Können Sie mit deutlich weniger Speicher oder Laufzeit auskommen, wenn Sie dafür deutlich mehr der jeweils anderen Ressource einsetzen? Wie? Geht es auch, ohne deutlich mehr einzusetzen?
- Können Sie das Assoziativgesetz ausnutzen, um die Berechnung von `hashCode` in `SongTree1` einfacher zu gestalten? Wenn ja, wie?
- Wenn Ihre Hashfunktion die Hashwerte zweier Songs mit einer Operation zu einem neuen Hashwert macht, für die das Assoziativgesetz nicht gilt, können Sie `hashCode` in `SongTree1` trotzdem implementieren, ohne eine Hilfsdatenstruktur aufzubauen? Wenn ja, wie? (Nicht ganz leicht: wenn Ihnen nichts einfällt, wird ihnen nichts abgezogen, aber denken Sie trotzdem darüber nach).