



## Übung zur Vorlesung *Einführung in die Informatik 2 für Ingenieure (MSE)*

Alexander van Renen (renen@in.tum.de)

<http://db.in.tum.de/teaching/ss20/ei2/>

### Blatt Nr. 6

Dieses Blatt wird am Montag, den 1. July 2020 besprochen.

### Teil 1: Objektorientierte Modellierung (in UML) und Programmierung (in Java)

Der erste Teil der Vorlesung zu Java und UML ist mit diesem Übungsblatt abgeschlossen. Jetzt ist also *die* Gelegenheit, den Stoff zu wiederholen. Dies erspart Ihnen nicht nur Lernaufwand am Ende des Semesters, sondern Sie haben außerdem noch die Möglichkeit in dieser letzten Übungsstunde zu Java offene Fragen zu klären. Zur Inspiration diese (unvollständige) Liste:

- UML
  - Klassendiagramme, Objektdiagramme
  - Assoziationen (Multiplizitäten, Aggregation vs. Komposition, Navigation)
  - Operationen (Konstruktoren, Beobachterfunktionen, Mutatoren, Access Modifier)
  - Vererbung (Generalisierung, Spezialisierung)
- Java
  - Umsetzung von UML in Java
  - Gemeinsame Unterobjekte
  - Typisierung
  - Garbage Collection
  - Werte vs. Objekte
  - Klassenattribute vs. Objektattribute
  - Methoden-Überladung
  - Vererbung und dynamisches Binden
  - Rekursion
  - Java Collections Framework (Set, List, Queue, Map)
  - Generische Typen
  - Iteration (for each loop, Iteratoren)
- Datenstrukturen
  - Komplexitätsangaben für Operationen ( $\mathcal{O}(1)$ ,  $\mathcal{O}(\log n)$ ,  $\mathcal{O}(n)$ )
  - Menge (Set), Verkettete Liste (LinkedList), Keller (Stack)
  - Binäre Suchbäume (Degenerierung, AVL-Baum)
  - Hashing (Double Hashing, Hashing with Chaining, Hashing with Linear Probing)

## Aufgabe 1: Java Garbage Collection

Der folgende Java-Programmcode generiert mehrere Objekte. Zwei der erzeugten Objekte werden am Ende des Programms nicht mehr referenziert und können daher von der *Garbage Collection* bereinigt werden. Welche sind dies? Geben Sie auch jeweils die Zeile an, ab der das Objekt nicht mehr referenziert wird.

```
1 Assistent wittgenstein = new Assistent(3004, "Wittgenstein",
2                                     "Sprachtheorie", null);
3 wittgenstein.boss = new Professor(2137, "Kant", Professor.Rang.C4);
4 Vorlesung ethik = new Vorlesung(5043, "Ethik", 3, wittgenstein.boss);
5 ethik.dozent = new Professor(2126, "Russel", Professor.Rang.C4);
6 wittgenstein.boss = new Professor(2133, "Popper", Professor.Rang.C3);
7 Student jonas = new Student(25403, "Jonas", 12);
8 Pruefung pruefung = new Pruefung(jonas, ethik,
9                                 wittgenstein.boss, termin);
10 pruefung.student = new Student(28106, "Carnap", 3);
11 pruefung.student = jonas;
12 wittgenstein.boss = new Professor(2136, "Curie", Professor.Rang.C4);
```

## Aufgabe 2: Komplexitätsangaben

Sie haben in der Vorlesung und Übung Komplexitätsangaben in der Landau-Notation (z.B.  $\mathcal{O}(n)$ ) kennen gelernt. Diese geben das asymptotische Laufzeitverhalten von Funktionen an. In dieser Aufgabe wollen wir feststellen, was dies in der Praxis bedeutet. Dafür messen wir die Laufzeit für das Nachschlagen in `HashMap` und `TreeMap`. Welche Laufzeitkomplexität erwarten Sie jeweils in Abhängigkeit von der Eingabegröße und können Sie diese mit Ihren Messergebnissen nachweisen?

### Tipps zur Messung

Fügen Sie zuerst die Elemente mit `put()` in die Abbildung ein und führen Sie dann 10 Millionen Lookups mit `get()` durch (so viele damit es messbar wird). Die Laufzeit können Sie messen, in dem Sie vor und nach den 10 Millionen Lookups mit `System.currentTimeMillis()` die aktuelle Zeit in Millisekunden seit 1970 abfragen. Die Differenz der beiden Zahlen ergibt entsprechend die Laufzeit in Millisekunden. Anschließend können Sie sich das Ergebnis z.B. in einem Punktediagramm in einem Tabellenprogramm Ihrer Wahl<sup>1</sup> veranschaulichen.

Wenn Sie überprüfen wollen, ob eine Laufzeit logarithmisch ist, sollten Sie die Messungen mit exponentiell ansteigender Eingabegröße durchführen (z.B. mit 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, ..., 1048576 Elementen). Wenn Sie die x-Achse Ihres Diagramms logarithmisch skalieren, wäre diese Laufzeit dann als Gerade erkennbar.

## Aufgabe 3: AVL-Bäume

Fügen Sie in einen AVL-Baum nacheinander die folgenden Elemente ein und führen Sie dabei die notwendigen Rotationen durch: 4, 8, 16, 12, 14, 3, 2, 6, 5

---

<sup>1</sup>Microsoft Excel, Google Docs, Apple Numbers, Open Office Calc, ...