



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

Christoph Anneser (anneser@in.tum.de)

<http://db.in.tum.de/teaching/ss23/ei2/>

Blatt Nr. 7

Dieses Blatt wird am Montag, den 19.06.2023 besprochen.

Aufgabe 1: Rekursion mit Boxen

Gegeben sei eine Menge an Boxen. Jede Box hat eine Länge, eine Höhe und eine Breite. Sie sollen nun den höchst-möglichen Stapel an Boxen bauen bzw. herausfinden, wie hoch maximal gestapelt werden kann. Dabei muss jedoch beachtet werden, dass eine Box b_1 nur dann auf eine Box b_2 gestellt werden kann, wenn Länge, Höhe und Breite von b_1 kleiner sind als die von b_2 .

Aufgabe: Vervollständigen Sie den untenstehenden Code an den mit `todo` gekennzeichneten Stellen.

Optional: Wir berechnen damit jedoch für manche Boxen die maximale Höhe mehrmals, wie können wir dies umgehen? Tipp: Verwenden Sie *Dynamische Programmierung*.

```
class Box implements Comparable<Box> {
    public Box(int id, int width, int heigth, int length) {
        this.id = id;
        this.width = width;
        this.height = heigth;
        this.length = length;
    }

    protected int id;
    protected int width;
    protected int height;
    protected int length;

    @Override
    public int compareTo(Box o) {
        // todo
    }
}

public class Main {
    static int getMaxHeigth(Set<Box> remainingBoxes, Box lastBox) {
        if (remainingBoxes.isEmpty())
            return lastBox != null ? lastBox.height : 0;
        // todo
    }
}
```

```

public static void main(String[] args) {
    Set<Box> boxes = new HashSet<>();
    boxes.add(new Box(1, 1, 2, 3));
    boxes.add(new Box(2, 4, 3, 3));
    boxes.add(new Box(3, 2, 2, 5));
    boxes.add(new Box(4, 1, 2, 1));
    boxes.add(new Box(5, 5, 5, 4));

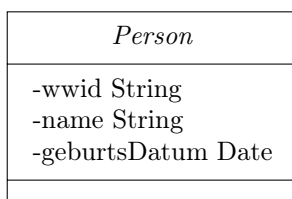
    int maxHeigth = getMaxHeigth(boxes, null);
    System.out.println(maxHeigth);
}
}

```

Aufgabe 2: UML Fußball Saison

- In einer Fußball-Saison gibt es 34 Spieltage. An jedem Spieltag finden genau 9 Spiele statt. Spieltage werden in aufsteigender Reihenfolge nummeriert.
- Es treffen in jedem Spiel genau zwei Mannschaften aufeinander. Jedes Spiel wird von vier Schiedsrichtern geleitet.
- Jede Mannschaft hat einen Namen, besteht aus mindestens 11, aber maximal 20 Spielern, und hat genau einen Trainer.
- Für die Ereignisse Tor und Karte muss sowohl die Spielminute als auch der beteiligte Spieler gespeichert werden. Außerdem muss der Typ des Tors (z.B. Elfmeter oder Eigentor) und die Farbe der Karte gespeichert werden.
- Für alle Schiedsrichter, Trainer und Spieler wird der Name, das Geburtsdatum sowie eine eindeutige Identifikationsnummer (WWID) im System gespeichert. *Die abstrakte Klasse Person ist bereits gegeben.* Für jeden Schiedsrichter wird zudem die Anzahl bereits geleiteter Spiele gespeichert. Jeder Trainer hat eine bestimmte Lizenz, z.B. A, B, und C. Ein Spieler hat eine Stammposition, z.B. Torwart oder Stürmer.

Ergänzen Sie das folgende UML-Klassendiagramm, in dem bereits die abstrakte Klasse Person gegeben ist, um die noch fehlenden Klassen und Assoziationen, um diesen Sachverhalt strukturell sinnvoll darzustellen und Redundanz so weit wie möglich zu vermeiden. Führen Sie passende Attribute ein. Verwenden Sie entsprechende Assoziationen und Multiplizitäten, falls nötig. Sichtbarkeiten (Modifier) der Attribute müssen *nicht* explizit angegeben werden.



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 3: Motivation von DBMS

Nennen Sie drei typische Probleme, die bei dem Verzicht auf ein Datenbankverwaltungssystem eintreten können. Überlegen Sie sich jeweils ein Beispiel bei dem das Problem auftritt.

Aufgabe 4: Terminologie

Beschreiben Sie die folgenden Begriffe der relationalen Modellierung. Verwenden Sie die Relation Professoren aus Abbildung 1 um Beispiele für die einzelnen Konzepte anzugeben:

Attribut • Schlüssel • Relation • Domäne • Tupel • Schema • Ausprägung

| Professoren | | | |
|-------------|------------|------|------|
| PersNr | Name | Rang | Raum |
| 2125 | Sokrates | C4 | 226 |
| 2126 | Russel | C4 | 232 |
| 2127 | Kopernikus | C3 | 310 |
| 2133 | Popper | C3 | 52 |
| 2134 | Augustinus | C3 | 309 |
| 2136 | Curie | C4 | 36 |
| 2137 | Kant | C4 | 7 |

Abbildung 1: Professoren in der relationalen Modellierung