**TU Berlin**
**Software and Business Engineering**
**Prof. Dr. Ingo Weber**
**Dr. Tobias Heindel**
**Richard Hobeck**
**Martin Grambow**

**WiSe 2020/21**
**Aufgabe 9**
**Block 3**
**18. Januar 2021**

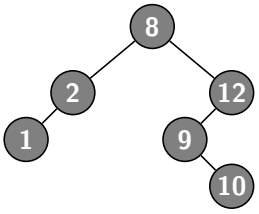
**Abgabetermin: 24. Januar 2021**

# Aufgabe 9 *Einführung in die Programmierung mit Java*

**Hinweise**

- Die Abgabe dieser Übungsaufgaben muss bis spätestens Sonntag, den 24. Januar 2021 um 23:59 Uhr im ISIS-Kurs erfolgt sein. Es gelten die Ihnen bekannten Übungsbedingungen.
- Lösungen zu diesen Aufgaben sind als gezippter Projektordner abzugeben. Eine Anleitung zum Zippen von Projekten finden Sie auf der Seite des ISIS-Kurses. *Bitte benutzen Sie einen Dateinamen der Form* `VornameNachname.zip`.
- Bitte beachten Sie, dass Abgaben im Rahmen der Übungsleistung für die Zulassung zur Klausur relevant sind. Durch Plagiieren verwirken Sie sich die Möglichkeit zur Zulassung zur Klausur in diesem Semester.

```
import java.util.LinkedList; import java.util.List;

public class SortedBinTree <T extends Comparable>{
    Node<T> root;

    public SortedBinTree(){
        root = null;
    }

    public void insertInOder(T v){/*...*/}


    public List<T> toOrderedList(){/*...*/}


    public List<T> getNodesOnLevel(int i){/*...*/}

}

class Node<T>{
    T val;
    Node<T> left;
    Node<T> right;

    Node(T v){
        val = v;
        left = null;
        right = null;
    }
}
```



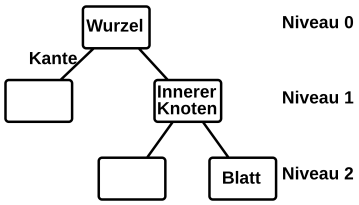




Abbildung 1: Operationen für (sortierte) Binärbäume

## Aufgabe 9.1 *Operationen für (sortierte) Binärbäume* (4 Punkte)

Erstellen Sie die Klasse `SortedBinTree` wie in Abbildung 1. Vervollständigen Sie die Methoden

`insertInOrder` für das Einfügen neuer Elemente in den Baum unter Erhaltung der Sortierung

`toOrderedList` für die Überführung in eine geordnete Liste aller Elemente und

`getNodesOnLevel` für das Auflisten der Elemente eines gegebenen Niveaus im Baum.

Der Baum oben rechts in Abbildung 1 resultiert also aus Einfügen der Elemente $8, 2, 1, 12, 9, 10$ (in dieser Reihenfolge), die sortierte Liste aller Elemente ist $1, 2, 8, 9, 10, 12$ und die Elemente auf Nieveau 1 sind 2 und 12.
Schließlich, fügen Sie eine `main`-Methode hinzu, in der sie alle drei Methoden geeignet testen.

```
import java.util.PriorityQueue;

public class HeapSort {

    public static void heapSort(Comparable[] a){
        PriorityQueue<Comparable> x = new PriorityQueue<Comparable>();
//
    }

}
```

Abbildung 2: Heapsort per Priority Queue

**Aufgabe 9.2** *Heapsort per Priority Queue* **(1 Punkt)**

Machen Sie sich mit der Klasse `PriorityQueue` vertraut.[1] Erstellen Sie die Klasse `HeapSort` wie in Abbildung 2. Implementieren Sie dann `heapSort` durch wiederholtes Aufrufen der Methode `poll` des `PriorityQueue`-Objekts (nach Überführen des Formalparameters in ein `PriorityQueue`-Objekt).

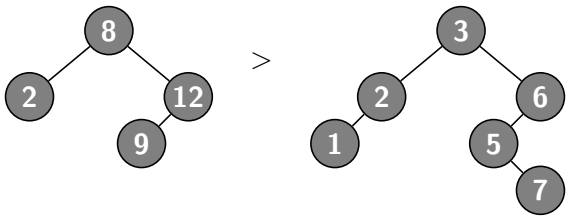
**Hinweis** In der Tat ist dies eine Variante von HeapSort; allerdings brauchen Sie dieses Wissen nicht für diese Aufgabe.

```
import java.util.List;

public class ComparableSortedBinTree<T extends Number> extends SortedBinTree implements Comparable<ComparableSortedBinTree<T>>{

    @Override
    public int compareTo(ComparableSortedBinTree<T> csbt) {/*...*/}

    public double sum(){/*...*/}
}
```

Abbildung 3: Bäume Vergleichbar machen

**Aufgabe 9.3** *Vergleichen von Bäumen bzgl. der Summe aller Knoten* **(1 Punkt)**

Erstellen Sie die Klasse `ComparableSortedBinTree` wie in Abbildung 3. Vervollständigen Sie die Methoden `compareTo` und `sum`, sodass ein Objekte dieser Klasse kleiner ist als ein gegebenes falls die Summer aller Zahlen im Baum kleiner ist als die des gegebenen Objekts.




Zum Beispiel ist der linke der oben abgebildeten Bäume größer als der rechte, da $31 > 24$.

[1] `java.util.PriorityQueue`