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

WiSe 2020/21
Aufgabe 8
Block 3
11. Januar 2021

Abgabetermin: 17. Januar 2021

Aufgabe 8 *Einführung in die Programmierung mit Java*

Hinweise

- Die Abgabe dieser Übungsaufgaben muss bis spätestens Sonntag, den 17. 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.

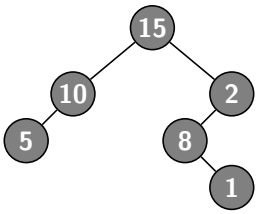


Abbildung 1: Ein Baum von Zahlen

```
public class Tree<T> {
    protected Node<T> root = null;

    public Tree(Node node) { root = node; }

    public Tree() { }

    public String toString(){
        if (root != null){
            return "["+root.toString()+"]";
        } else {
            return "ø";
        }
    }
}
```

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

    Node(T v, Node l, Node r){
        val = v;
        left = l;
        right = r;
    }

    public String toString(){
//
    }

}
```

Abbildung 2: Baum-Klasse in Java

Aufgabe 8.1 *Bäume als Zeichenketten* (2 Punkte)

Vervollständigen Sie die `toString`-Methode der Klasse `Node`, sodass ein Baum wie in Abbildung 1 wie folgt ausgegeben würde:

`[15:[10:[5:ø,ø],ø],[2:[8:ø,[1:ø,ø]],ø]]`

Fügen Sie der Klasse `Tree` eine `main`-Methode hinzu, die einen Baum wie in Abbildung 1 erzeugt und anschließend mittels der `toString`-Methode ausgibt.

```
public class NumberTree<T extends Number> extends Tree{

    public double weight() {
        if (root == null) return 0;
        double res = 0;
        //
        return res;
    }

    NumberTree(Node<T> n){
        super();
        root  = n;
    }

}
```

Abbildung 3: Zahlen-Bäume in Java

Aufgabe 8.2 *Bäume von Zahlenwerten und ihr Gewicht* **(1 Punkt)**

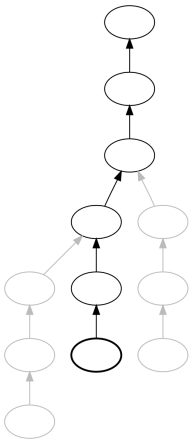
Erstellen Sie die Klasse `NumberTree` wie in Abbildung 3. Vervollständigen Sie die Implementierung der Methode `weight`, die das *Gewicht* des Baumes zurückgeben soll, also die Summe aller Zahlen im Baum. Für das Beispiel in Abbildung 1 ist das Gewicht also 41. Fügen Sie eine `main`-Methode hinzu, in der Sie das Gewicht des Baumes aus Abbildung 1 berechnen.

Tipp Die Methode `Number.doubleValue()` liefert den `double`-Wert eines `Number`-Objekts.

Aufgabe 8.3 *Menge der Blätter im Baum* **(1 Punkt)**

Fügen Sie Ihrer Klasse `Tree` eine Methode `getLeaves` ohne Formalparameter hinzu, welche die Menge der Blätter in einem Baum zurück gibt, und zwar als `java.util.HashSet<Node<T>>`.

Erinnerung Ein Blatt ist ein Knoten, dessen linkes und rechtes Kind beide `null` sind.



```
class List<T>{
    ListEl<T> head = null;

    // feel free to add methods

    List(){}

    List(ListEl<T> el){
        head = el;
    }
}
```

```
class ListEl<T> {
    ListEl<T> next;
    T val;

    // feel free to add methods

     ListEl(ListEl n, T v){
        next = n;
        val = v;
    }
}
```

Abbildung 4: Ungerichteter Baum (auch als ≫Spaghetti-Stack≪ bekannt); Listen in Java

Aufgabe 8.4 *`HashSets` von Listen = ungerichtete Wälder ⊇ ungerichtete Bäume* **(2 Punkte)**

Erstellen Sie eine Klasse `UndirectedForest` mit Typ-Paramter `T`, die eine Objekt-Variable vom Typ `HashSet<List<T>>` hat. Implementieren Sie eine Methode `getRoots`, ohne Formalparameter, die ein `HashSet<Node<T>>` zurück gibt, welches der *Menge* der *letzten* Elemente aller Listen entspricht.

Tipp Schreiben Sie zuerst eine Methode, die das letzte `ListEl`-Objekt einer Liste ermittelt.

Hinweis Wenn alle Listen den gleichen letzten Knoten haben, dann besteht der Wald aus genau einem Baum.

```
public class QuickSort {

    public static void quickSort(int[] a){
        quickSortRange(a,0,a.length-1);
    }

    private static void swap(int[]a, int i, int j ){
        int tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
    }

    private static void quickSortRange(int[] a,int i,int j){
        assert(i <= j) : "i>j last index is before first index "+i+","+j+" -- no good!";

        int length = j-i+1;
        switch (length){
            case 1: return;
            case 2: if (a[i] > a[j]){
                    swap(a,i,j);
                }
                return;
            default: break;
        }

        int middle = i+(length/2);

        int pivot = a[middle];
        // move occurrence of the pivot value to the front
        swap(a,i,middle);

        // _divide_ the portion of the array
        int l = i+1;
        int r = j;
        while(l<r) {
            while (a[l] <= pivot && l < r) {
                l++;
            }
            while (a[r] >= pivot && l < r) {
                r--;
            }
            assert (l==r || (a[l] > pivot && a[r]<pivot));
            if (l!= r){
                swap(a,l,r);
            }
        }
        assert(l==r);

        // there is this special case to take care of
        if (a[l] > pivot ){ l--; }

        // move pivot in the right spot ...
        swap(a,i,l);

        // ... and  _conquer_
        assert(l>=i); if(l>i)quickSortRange(a,i,l-1);
        assert(l<=j); if(l<j)quickSortRange(a,l+1,j);
    }

    public static void quicksort(int[] a){
        quickSortRange(a,0,a.length-1);
    }
}
```

Abbildung 5: Quicksort

Aufgabe 8.5 *Quicksort und der Baum der Pivot-Elemente* **(2 Punkte)**

Adaptieren Sie die Implementierung in Abbildung 5, sodass die Methode `quicksort` ein Objekt des Typs `Tree<Integer>` zurück gibt, welches ein sortierter Baum ist, dessen inneren Knoten den *Pivot*-Elementen entsprechen—wobei die *Pivot*-Elemente eben diejenigen `int`-Werte sind, die in der Zeile

```
int pivot = a[middle];
```

zugewiesen werden.

Aufgabe 8.6 *Tree parsing* ***(1 Zusatzpunkt)***

Fügen Sie Ihrer `Tree`-Klasse eine Klassen-Methode `parseTree` hinzu, mit einer Zeichenkette als Formalparameter. Die Methode soll jeden Baum aus seiner `String`-Repräsentation rekonstruieren. Die Rückgabe ist der entsprechenden Baum als `Tree<Integer>`-Objekt. Zum Beispiel, wenn die Eingabe `[15:[10:[5:ø,ø],ø],[2:[8:ø,[1:ø,ø]],ø]]` ist, dann soll der Baum aus Abbildung 1 zurückgegeben werden.

Tipps

(a) `Integer.parseInt(s,i,j,10)` versucht die Zeichen von Index i bis Index $j-1$ im String s als Integer zu interpretieren (zur Basis 10); also ein Aufruf von

```
Integer.parseInt(s,k,k,10)
```

ergibt immer einen Fehler und

```
Integer.parseInt(s,k,k+1,10)
```

interpretiert das Zeichen an Index k als Ganzzahl.

(b) `java.util.Stack` implementiert Stacks.