

Prüfung vom 8. April 2019: Teil 1: 18 Punkte

Name: _____

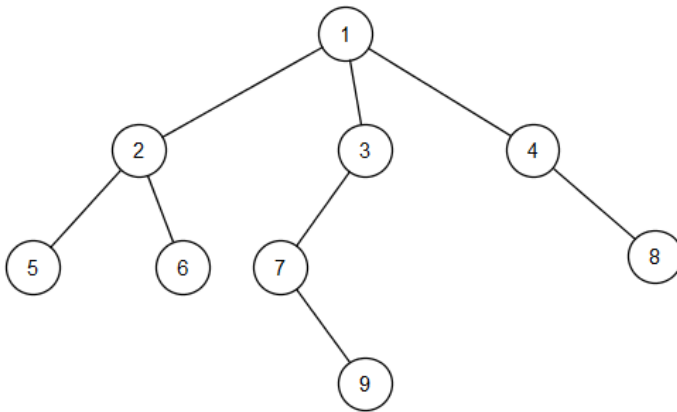
Allgemeine Hinweise für Teil 1:

- 1) Erlaubte Hilfsmittel: 1 A4 Blatt (2 A4 Seiten) Unterlagen in ausgedruckter oder handgeschriebener Form.
- 2) Nicht erlaubte Hilfsmittel: Elektronische Geräte jeglicher Art (auch kein Handy, Taschenrechner), Kommunikation mit anderen Personen.
- 3) Schreiben Sie die Antworten direkt auf das Aufgabenblatt.
- 4) Bewertet werden die Korrektheit der Resultate sowie die Herleitung / Begründung.

Viel Erfolg!

Aufgabe 1: Bäume

(3 + 2 = 5 Punkte)



a) Beantworten Sie die folgenden Fragen zum oben abgebildeten Baum:

1. Besuchen Sie alle Knoten mit einer *Post-Order-Traversierung* und geben Sie die besuchten Knoten in entsprechender Reihenfolge aus.
2. Markieren Sie alle Knoten auf dem Niveau 3.
3. Welches ist die Ordnung des abgebildeten Baumes mindestens? Begründen Sie!
4. Bei welchen Knoten handelt es sich um innere Knoten?
5. Ist der Baum ausgefüllt? Begründen Sie Ihre Antwort.

b) Zeichnen Sie einen Binärbaum mit den folgenden Eigenschaften:

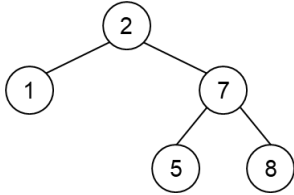
- Jeder Knoten speichert einen Buchstaben.
- Die Preorder-Reihenfolge der Knoten des Baumes ist: BERTSCHI
- Die Inorder-Reihenfolge lautet: ETRBSHCI

Aufgabe 2: Operationen auf Binären Suchbäumen

(2 + 1 + 2 = 5 Punkte)

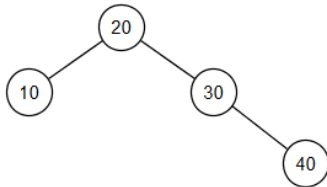
- a) In den gezeichneten *AVL-Baum* wird die Zahl 3 eingefügt. Zeichnen Sie den neuen Knoten ein. Machen Sie eine neue Zeichnung, für jeden Schritt, bei dem der Baum von Ausgleichsoperationen verändert wird.

add(3):

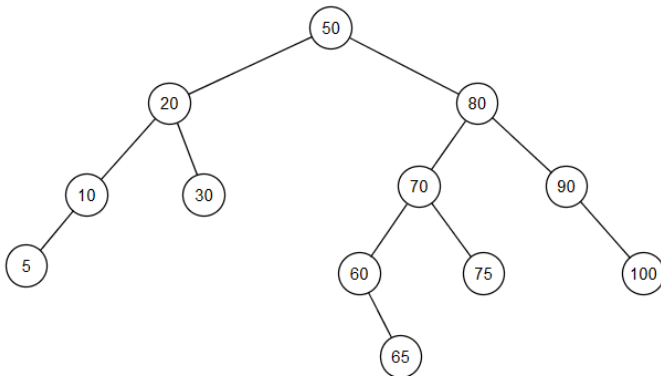


- b) In den gezeichneten *AVL-Baum* werden die angegebenen Zahlen in entsprechender Reihenfolge eingefügt. Zeichnen Sie die neuen Knoten ein. Machen Sie eine neue Zeichnung, für jeden Schritt, bei dem der Baum von Ausgleichsoperationen verändert wird.

add(80); add(90):



- c) Wie sieht der nachfolgende AVL-Baum nach dem Entfernen des Knotens 30 aus? Korrekturen können auch in den Original-Baum vorgenommen werden, solange die Operationen nachvollziehbar bleiben.

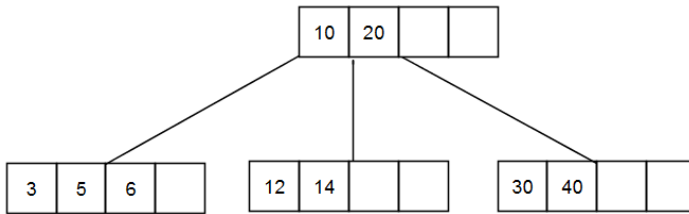


Aufgabe 3: B-Bäume

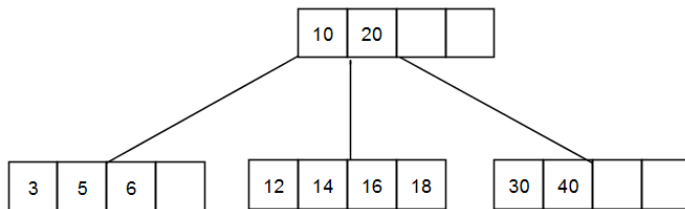
(4 Punkte)

Führen Sie die unten aufgeführten Operationen in der angegebenen Reihenfolge auf dem abgebildeten B-Baum aus. Zeichnen Sie den kompletten Baum neu, wenn es der Übersicht dient.

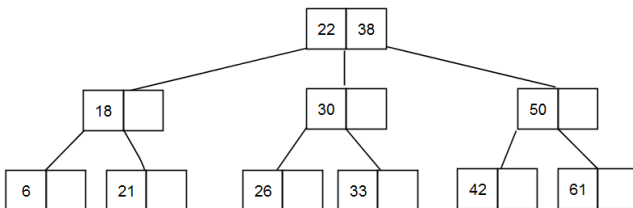
a) B-Baum der **Ordnung 2**: Add(1); Delete(40)



b) B-Baum der **Ordnung 2**: Add (11)



c) B-Baum der **Ordnung 1**: Delete(18)



Aufgabe 4: SpecialMethod auf der einfach verketteten Liste

(1 + 2 + 1 = 4 Punkte)

Sie haben eine einfach verkettete Liste ohne last-Zeiger. Null kann nicht als Element in der Liste abgelegt werden. Gegeben sind folgende Klassen-Definitionen:

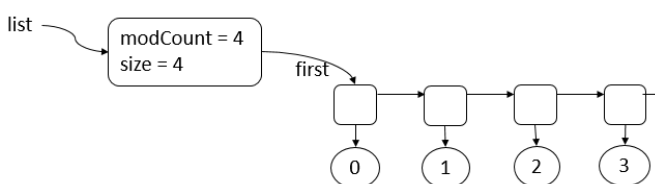
```
public class SinglyLinkedList<E> implements List<E> {
    private int size = 0, modCount = 0;
    private Node<E> first;
    ...
    private static class Node<E> {
        private final E elem;
        private Node<E> next;
        ...
    }
}
```

In der Klasse SinglyLinkedList befindet sich folgende Methode:

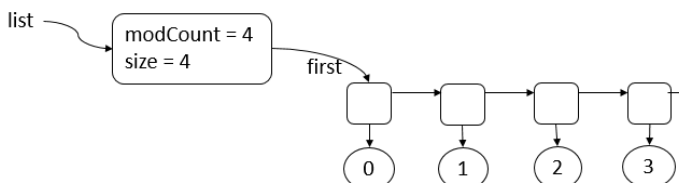
```
public boolean specialMethod(Object elem) {
    Node<E> n = first, p = null;
    while (n != null && !n.elem.equals(elem)) {
        p = n;
        n = n.next;
    }
    if (n != null) {
        if (p != null) {
            p.next = n.next;
            n.next = first;
            first = n;
        }
        return true;
    } else {return false;}
}
```

Beantworten Sie folgende Fragen:

a.) Gibt der Aufruf specialMethod(0) auf der nachfolgenden Liste true oder false zurück?



b.) Zeichnen Sie die Liste, nachdem darauf specialMethod(2) aufgerufen wurde:




c.) Die Methode specialMethod in der oben stehenden Implementation verändert den Modification Count der Liste nie. Ist das korrekt? Begründen Sie.

Prüfung vom 8. April 2019: Teil 2: 22 Punkte

Name: _____

Allgemeine Hinweise für Teil 2:

- 1.) Erlaubte Hilfsmittel: 1 A4 Blatt (= 2 A4 Seiten) Unterlagen in ausgedruckter oder handgeschriebener Form (die gleichen wie im ersten Teil), Computer in der vorbereiteten Lernstick-Umgebung OHNE Zugriff auf lokale Speichermedien und Internet. Der Lernstick darf vor dem Start nicht verändert und auch nicht in einer VM gebootet werden.
- 2.) Betrugsversuche führen automatisch zur Note 1 und werden bei der Studiengangleitung sowie auch bei der Ausbildungsadministration gemeldet.
- 3.) Nicht erlaubte Hilfsmittel: Andere Elektronische Geräte (Handy, Taschenrechner), Kommunikation mit anderen Personen.
- 4.) Ändern Sie keine Visibility-Modifiers (private, public, default) oder final Keywords.
- 5.) Es gibt pro Aufgabe eine eigene Klasse, die zu erweitern ist. Den zu ergänzenden Code finden Sie jeweils am Ende nach der main-Methode. Die Main-Methode hat schon Beispiel-Code. Die zu ergänzenden Methoden werfen per Default eine UnsupportedOperationException. Ersetzen Sie diese durch Ihren Code (kompletter Code in einer Klasse).
- 6.) Die Dateien finden Sie in den Sub-Packages (1 Klasse pro Sub-Package) von: `ch.fhnw.algd2.exam.stud.mo`
- 7.) Im Sub-Package „common“ brauchen / dürfen keine Veränderungen vorgenommen werden. Dabei handelt es sich um Interfaces und abstrakte Klassen, die eine Dummy-Implementation für viele nicht gebrauchte Methoden bietet.
- 8.) Benennen Sie die Klassen jeder Aufgabe um, indem Sie „Stud“ durch ihren Vor- und Nachnamen (ohne Umlaute) in CamelCase ersetzen: Zum Beispiel: `MyLinkedList_Stud.java` => `MyLinkedList_MichaelMueller.java`.
- 9.) Lösen Sie die Programmieraufgaben direkt mit dem Lernstick in der IDE (Eclipse oder IntelliJ). Die Projekte sind da bereits importiert. Speichern Sie regelmässig. Falls sie auf den IDE Support verzichten möchten (nicht empfohlen), können Sie auch LibreOffice verwenden.
- 10.) Nach der regulären Prüfungszeit bekommen Sie noch zusätzlich Zeit, um die Antworten vom Computer auf das Aufgabenblatt abschreiben zu können. In dieser Zeit dürfen keine Veränderungen an den Dateien mehr vorgenommen werden (Tippen auf der Tastatur ist verboten).
- 11.) Die auf das Aufgabenblatt übertragenen Lösungen dienen als Backup. Korrigiert werden schlussendlich die Dateien.
- 12.) Mit der Beschriftung der Prüfung bestätigen Sie zugleich, dass die Lösungen auf dem Aufgabenblatt und auf dem Lernstick deckungsgleich sind.
- 13.) Fahren Sie den Lernstick sauber über das Menü oben rechts und den Ausschalt-Knopf herunter:  (kann bis zu einigen Minuten dauern).

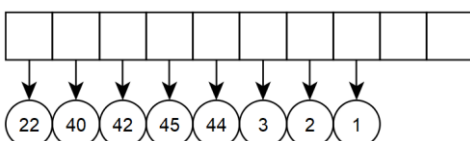
Viel Erfolg!

Aufgabe 5: Iterator auf der Pyramiden-Collection

(6 Punkte)

Die Klasse `PyramidCollection` (Sub-Package: `collection`) kann nur Elemente aufnehmen, welche das `Comparable` Interface implementieren. Die Collection hat die Eigenschaft, dass die Elemente pyramidenförmig angeordnet sind. Dies bedeutet, dass die Elemente von den Enden ausgehend gegen die Mitte aufsteigend sortiert sind (anhand der `Comparable`-Definition). Die Elemente selbst werden innerhalb der `PyramidCollection` Klasse aus einem Array referenziert.

Ein Beispiel für eine gültige Anordnung von 8 Integer-Werten im Array der `PyramidCollection` wäre:



Die zugehörige Klassendefinition ist:

```
public class PyramidCollection<E extends Comparable<?
super E>> implements Collection<E> {

    private E[] data;
    private int size;
    private int modCount;
    ...
}
```

Gehen Sie davon aus, dass die `PyramidCollection` das `Collection` Interface korrekt implementiert (und den `Modification-Count` wenn nötig erhöht). Der `Iterator` fehlt jedoch noch. Implementieren Sie als innere Elementklasse mit dem Namen `PyramidIterator` einen `Iterator` (ohne die `remove`-Methode).

- Der `Iterator` soll alle Elemente **in aufsteigend sortierter Reihenfolge** (anhand der `Comparable` Definition) zurückliefern.
- Ein Aufruf der `next()` Methode muss eine `IllegalStateException` werfen, wenn sich die `Collection` verändert hat.
- Der Aufruf der `next()` Methode muss eine `NoSuchElementException` werfen, wenn keine weiteren Elemente mehr zurückgegeben werden können.
- `Null`-Werte können nicht in die `Collection` hinzugefügt werden.

Implementations-Regeln:

- Benennen Sie die Klasse `PyramidCollection_Stud.java` um. (Ersetzen Sie das „Stud“ mit Ihrem Vor- und Nachnamen (ohne Umlaute), z.B. `PyramidCollection_MichaelMueller.java`).
- Sie dürfen im `Iterator` **KEINE** Methoden der `PyramidCollection` oder anderer Klassen verwenden (Variablen der `PyramidCollection`-Klasse jedoch schon). Ausnahmen sind natürlich die notwendigen `Exception`-Klassen.

Implementieren Sie den `Iterator` mit den Methoden `hasNext()` und `next()` wie zuvor definiert:

```
private class PyramidIterator implements Iterator<E> {
```

```
}}
```

Aufgabe 6: Prüfen, ob ein Binärer Suchbaum ausgefüllt ist**(4 Punkte)**

Gegeben ist ein Binärer Suchbaum in der Klasse `TreeStructure` (Sub-Package: `tree`), dessen Nodes jedoch zur Vereinfachung nur `int`-Werte als Schlüssel abspeichern können (die Schlüssel sind für diese Aufgabe irrelevant). Der Wurzelknoten ist über die `root`-Referenz erreichbar. Die Klassendefinition ist wie folgt:

```
public class TreeStructure implements TreeBalance {
    private Node root = null;

    private static class Node {
        private Node left;
        private Node right;

        private final int key;

        private Node(int key) {
            this.key = key;
        }
    }
    ...
}
```

Implementieren Sie innerhalb der Klasse `TreeStructure` eine Methode `boolean isAusgefuehlt()`, die zurückgibt, ob der Binäre Suchbaum ausgefüllt ist. Ein leerer Suchbaum gilt als ausgefüllt.

Implementations-Regeln:

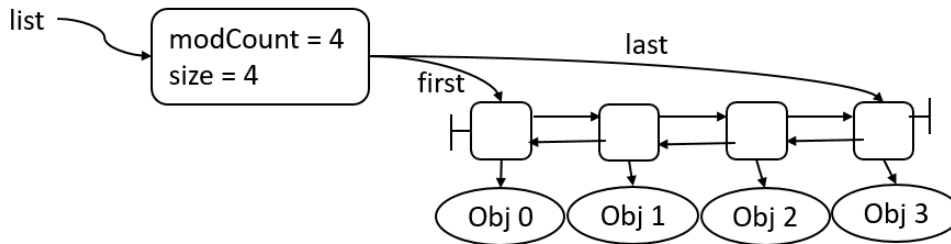
- Benennen Sie die Klasse `TreeStructure_Stud.java` um. (Ersetzen Sie das „Stud“ mit Ihrem Vor- und Nachnamen (ohne Umlaute), z.B. `TreeStructure_MichaelMueller.java`).
- Es dürfen keine Hilfsmethoden externer Klassen verwendet werden.

```
public boolean isAusgefuehlt() {
```


Aufgabe 7: Swap-With-Previous Funktion auf doppelt verketteter Liste

(2 + 10 = 12 Punkte)

Vorhanden ist eine doppelt verkettete Liste (ohne Dummy-Head), wie sie aus dem Unterricht bekannt ist:



Sie können davon ausgehen, dass die Liste das Collection Interface korrekt implementiert. Die Klassendefinition der `DoublyLinkedList` (Sub-Package: `list`) ist:

```
public class DoublyLinkedList <E extends Comparable<? super E>> implements List<E> {
    private int size;
    private int modCount;

    private Node<E> first;
    private Node<E> last;

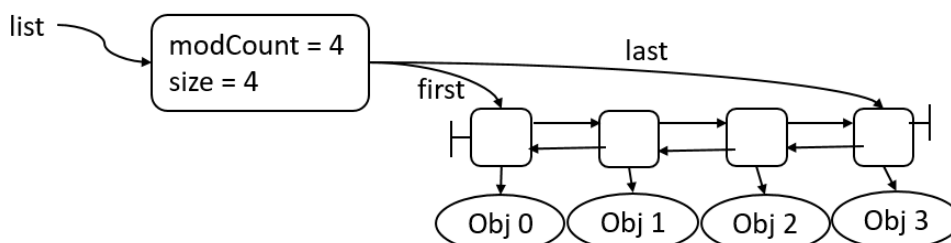
    ...
}
```

Die `Node`-Klasse mit Referenzen auf den vorherigen und den nächsten `Node` innerhalb der Liste ist wie folgt definiert:

```
public class Node<E> {
    public final E elem;
    public Node<E> next;
    public Node<E> prev;
    ...
}
```

Implementieren Sie eine Methode `boolean swapWithPrevious(int index)`, welche das Element am gegebenen Index mit dessen Vorgänger vertauscht. Werfen Sie eine `IndexOutOfBoundsException`, wenn ein Index angegeben wurde, dessen Element keinen Vorgänger besitzt oder sich an entsprechender Stelle kein Element befindet.

a.) Korrigieren Sie die Node-Referenzen in der unteren Abbildung, nachdem `swapWithPrevious(2)` aufgerufen wurde:



(Die Implementations-Aufgabe befindet sich auf der nachfolgenden Seite)

b.) Implementieren Sie die Methode `swapWithPrevious`

Implementations-Regeln & Hinweise

- Benennen Sie die Klasse `DoublyLinkedList_Stud.java` um. (Ersetzen Sie das „Stud“ mit Ihrem Vor- und Nachnamen (ohne Umlaute), z.B. `DoublyLinkedList_MichaelMueller.java`).
- Sie dürfen keine Methoden aus anderen Klassen oder aus der `DoublyLinkedList` verwenden. Ausnahmen sind natürlich die notwendigen Exception-Klassen.
- Die `elem` Variable der Klasse `Node` muss final bleiben.
- Denken Sie daran, dass Sie möglicherweise Instanz-Variablen der `DoublyLinkedList` anpassen müssen.

```
public void swapWithPrevious(int index) {
```