## Team:

5, Marc Kaepke & Constantin Wahl

## Aufgabenteilung:

M. Kaepke

Ausarbeitung der Skizze

C. Wahl

Messungen und Auswertung

Gemeinsam

Implementierung des ADTs

## Quellenangabe:

Vorlesungsscript und Folien

## Bearbeitungszeitraum:

Gemeinsam

M. Kaepke

2 Stunden: Skizze

0,5 Stunden: Count-Klasse und #sortNum() erweitert

C. Wahl

## Aktueller Stand:

Skizze wurde erstellt. Mit der Implementierung wurde bereits begonnen.

# **Skizze**

## Package-Struktur:

src.adt.implementations.

* AdtArrayImpl
* AdtAVLBaumImpl
* AdtListImpl

src.adt.interfaces.

* AdtArray
* AdtAVLBaum
* AdtList

src.general.

* NumGenerator
* Count

src.tests.

* AdtAVLBaumTests

## NumGenerator:

Die Methoden #sortNum(String, int) und #sortNum(String, int, boolean) werden jeweils um einen weiteren Parameter erweitert, der angibt, ob Duplikate beim Generieren erlaubt sind oder nicht.

#sortNum(String, int) 🡪 #sortNum(String, int, boolean)

#sortNum(String, int, boolean) 🡪 #sortNum(String, int, boolean, boolean)

## ADT-AVL-Baum:

Funktional (nach außen), Java Interface Syntax:

* public static AdtAVLBaum create()
  + Erzeugt eine neue Adt AVLBaum
* public boolean isEmpty()
  + Gibt zurück ob der AVLBaum leer ist
  + true => der Baum ist leer (hat keinen Knoten)
  + false => der Baum hat mindestens einen Knoten
* public int high()
  + Gibt die Gesamthöhe des Baums zurück
* public void insert(int elem)
  + Fügt den Integer-Wert in den Baum ein
* public long insertRunTime(int elem)
  + Fügt den Integer-Wert in den Baum ein
  + Misst die Laufzeit in Nanosekunden
* public Count insertCount(int elem)
  + Fügt den Integer-Wert in den Baum ein
  + Zählt die die Links- & Rechtsrotationen und die Lese- & Schreibzugriffe
  + Die Informationen sind im Return-Type Count gespeichert
* public void delete(int elem)
  + Löscht den Integer-Wert bzw. den Knoten aus dem Baum
  + Gibt keine Rückmeldung, wenn das Element nicht vorhanden ist
* public Count deleteCount(int elem)
  + Löscht den Integer-Wert bzw. den Knoten aus dem Baum
  + Zählt die die Links- & Rechtsrotationen und die Lese- & Schreibzugriffe
  + Die Informationen sind im Return-Type Count gespeichert
* public boolean print()
  + Exportiert den Baum als \*.png Datei

Technisch (nach innen):

* Beim Export wird als Zwischendatei eine \*.dot Datei erzeugt
* Beim Einfügen oder Löschen eines Knotens muss geprüft werden, ob der Baum nach der Aktion noch balanciert ist, andernfalls muss eine der Rotationsfunktionen aufgerufen werden

JUnit:

* Es muss überprüft werden, das Elemente nicht doppelt eingefügt werden können
* Es muss überprüft werden, ob das Entfernen des letzten Knotens dafür sorgt, das die #isEmpty() Methode mit true evaluiert
* Es muss überprüft werden, das jede Einfüge- und Löschoperation die Höhe anpasst
* Es muss überprüft werden, ob die ADT mit großen Datenmengen (bspw. 300 Knoten) einwandfrei funktioniert

## Count:

Hält die Zähldaten zu den Links- und Rechtsrotationen und den Lese- und Schreibzugriffen.

Konstruktor:

public Count(int, int, int, int)

Methoden:

Getter-Methoden für Instanzvariablen