## 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

3 Stunden: Skizze

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

C. Wahl

1 Stunde: Skizze

## Aktueller Stand:

Skizze wurde erstellt. Mit der Implementierung wurde bereits begonnen.

## Aktualisierung an der Skizze:

Die ADT ist detaillierter beschrieben. Dazu gehört der funktionale und technische Bereich, ebenso die Fehlerbehandlung, Objektmengen und die Operationen an der ADT.

# **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:

Ein AVL Baum ist ein balancierter binärer Suchbaum. Er ist genau dann balanciert, wenn sich für jeden Knoten die Höhe der zugehörigen Teilbäume um höchstens 1 unterscheidet.

Funktional (nach außen):

* Die Elemente sind vom Typ „ganze Zahlen“
* Beim Einfügen wird das neue Element gemäß der Sortierung als Knoten in den AVLBaum eingegliedert
  + Alle Knoten im linken Teilbaum < Wurzel
  + Alle Knoten im rechten Teilbaum > Wurzel
* Es lassen sich Elemente nicht doppelt einfügen (duplikatenfrei)
* Nach jeder Einfüge- oder Löschoperation ist der AVLBaum balanciert

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
* Die ADTAVLBaum ist intern mittels einer Datenstruktur „AVLKnoten“ zu realisieren

Objektmengen:

* elem (int), adt (AVLBaum), png

Operationen:

* 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
  + Ein leerer Baum hat die Höhe = 0
  + Ein Baum, nur mit der Wurzel ohne Kinder, hat die Höhe = 1
* public AdtAVLBaum insert(int elem)
  + Fügt das Element in den Baum ein und wird einsortiert
* public long insertRunTime(int elem)
  + Fügt das Element in den Baum ein
  + Misst die Laufzeit in Nanosekunden
* public Count insertCount(int elem)
  + Fügt das Element in den Baum ein
  + Zählt die Links- & Rechtsrotationen und die Lese- & Schreibzugriffe
  + Die Informationen sind im Return-Type Count gespeichert
* public AdtAVLBaum delete(int elem)
  + Entfernt das Element bzw. den Knoten aus dem Baum
* public Count deleteCount(int elem)
  + Entfernt das Element bzw. den Knoten aus dem Baum
  + Zählt die Links- & Rechtsrotationen und die Lese- & Schreibzugriffe
  + Die Informationen sind im Return-Type Count gespeichert
* public boolean print()
  + Exportiert den Baum als \*.png Datei

Fehlerbehandlung:

* Beim Einfügen eines ungültigen Knotens (z. B. keine ganze Zahl) wird das Einfügen des Knotens ignoriert.
* Wenn das einzufügende Element schon vorhanden ist wird die ADT nicht verändert.
* Soll ein nicht vorhandenes Element gelöscht werden, so wird die ADT unverändert zurückgegeben.

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