Aufgabenstellung

LinkedList

Iterator

- Vervollständige die Klasse `Iterator` für die Klasse LinkedList, indem du Funktionen `Iterator(ListElement<T> e)`, `boolean hasNext()`, `T next()` implementierst.

reverse

- Die Methode `void reverse()` soll die Liste umkehren, sodass das erste Element dann das Letzte ist, das zweite dann das Vorletzte usw..

contains:

- `boolean contains(T value)` soll zurückgeben, ob der Parameter `value` in der Liste vorhanden ist.

SortedList

Invariante

- Die Klasse SortedList ist eine Unterklasse von LinkedList und besitzt folgende Invariante: Alle Listenelemente sind aufsteigend sortiert.

insert

- Die Methode `void insert(T value)` soll `value` in die Liste hinzufügen, sodass die Invariante beibehalten bleibt.
- Hinweis: Da T mit `T extends Comparable<T>` eingrenzt wird, kannst du zum Vergleichen `int compareTo(T o)` verwenden.

TribonacciList

Für diese Aufgabe definieren wir die Tribonacci-Folge wie folgt:

$$a_n := \begin{cases} 1, & \text{falls } n \in \{0,1,2\} \\ a_{n-1} + a_{n-2} + a_{n-3}, & \text{falls } n \geq 3 \end{cases}$$

Die Liste soll nun die Folge repräsentieren. Beispielsweise könnte die Liste so aussehen:

[1] → [1] → [1] → [3] → [5] → [9]

addNext

`void addNext(int n)` soll die nächsten n Folgenglieder an die Liste anhängen. Würden wir für die Liste oben `addNext(3)` aufrufen, bekämen [1] → [1] → [1] → [3] → [5] → [9] → [17] → [31] → [57] als neue Liste.

StringList

Java speichert seine Strings im sog. **String Intern Pool** im Heap ab, sodass jeder String während der Laufzeit in diesem Pool auch nur einmal existiert. Manchmal will man dieses Verhalten nicht haben. Dazu gibt es Klassen wie StringBuilder und Stringbuffer.

Alternativ dazu kann man das auch mit verketteten Listen machen wie im Tutorium. Unsere Liste besitzt als `value` ein `char[]`. Das Aneinanderreihen (Konkatenieren) aller `char[]` repräsentiert dann unseren String.

Beispiel:

[tsch] → [üssik] → [o] → [ws] → [ki]

Unser String wäre demnach “tschüssikowski”.

Du darfst die vorgegebene Methode `String toString()` in dieser Aufgabe benutzen (auch wenn es bisschen sinnlos dann ist unseren String als Liste zu implementieren).

Es ist auch möglich das ohne die toString Methode zu verwenden, jedoch schwerer.

isPalindrom

- Diese Methode soll zurückgeben, ob die Liste welche unseren String repräsentiert, ein Palindrom ist.
- Ein Wortpalindrom ist ein Wort, das rückwärts gelesen dasselbe Wort ergibt bzw. es ist in der Mitte des Wortes gespiegelt.

reverse

- Die Methode soll die Liste so umkehren, sodass der repräsentierte String auch umgekehrt wird.. Deswegen überschreiben wir unsere Methode, da wir ihr nun eine andere Bedeutung geben.

  Beispiel: [tsch] → [üssik] → [o] → [ws] → [ki] $\Longrightarrow$ [ik]→[sw] → [o] → [kissü] →[hcst]

contains

- Wir überschreiben die Methode auch hier von der Oberklasse.. Diese soll nicht nun mehr überprüfen, ob der Parameter in eines der Listenelemente ist, sondern die Methode `boolean contains(char[] value)` soll ausgeben, ob `value` ein substring ist von unserer Liste.

NumberList

Diese Liste soll nun eine Zahl verkehrt herum repräsentieren. Beispielsweise würde die Zahl 54321 als Liste so aussehen: [1] → [2] → [3] → [4] → [5]. (Least significant digit als erstes Element).

Dabei gilt folgendes für alle `value` der Listenelemente: $0 <= value <= 9$. Außerdem soll die Liste keine führende Nullen besitzen.

insert

- `insert(Integer value)` soll `value` an die Liste anhängen. Dabei soll mit least significant digit angefangen werden (Zuerst Einerstelle, dann Zehnerstelle, dann Hunderterstelle usw.)

Beispiel: Wenn wir auf der Liste oben `insert(9876)` aufrufen, sieht unsere Liste dann so aus

[1] → [2] → [3] → [4] → [5] → [6] → [7] → [8] → [9]

Unsere List soll wie gesagt keine Zahl repräsentieren die führende Nullen hat.

crossSum

- Die Methode `Integer crossSum()` soll die Quersumme berechnen, indem sie alle Listenelemente addiert und zurückgibt.

add

- Die Methode `NumberList add(NumberList first, NumberList second)` nimmt als Eingabe zwei Listen und gibt eine neue Liste zurück, welche die Addition von first und second darstellen soll.
- Tipp: Schaue dir noch einmal an, wie schriftliches Addieren funktioniert. Nach einem ähnlichen Prinzip kannst du die Methode auch so implementieren.

Useful Links:

Recusion:

- https://www.youtube.com/watch?v=ngCos392W4w&

Linkedlist:

- https://www.youtube.com/watch?v=K1iu1kXkVoA&

Generics:

- https://www.youtube.com/watch?v=K1iu1kXkVoA

Iterator:

- https://www.youtube.com/watch?v=G3uNYHtn83c

String Pool (nicht wichtig):

- https://live.rbg.tum.de/w/EIDI/4000 1:39