Vorgehensweise und Entscheidungsfindung

Für die Speicherung der Elemente und zum Aufbau der Liste (Knoten) ist die innere, private Klasse Node. Diese enthält eine Objektreferenzvariable des generischen Typs der Listenklasse und weiterhin zwei Objektreferenzvariablen für den nächsten und vorherigen Knoten.
Zunächst haben wir überlegt, in der Listenklasse nur eine weitere Objektreferenzvariable (head) zu verwenden. Man könnte dann mit head.prev an den tail kommen, ohne eine weitere Variable zu haben. Laut Aufgabenstellung sollten wir aber head und teil nehmen.
Die Größe der Liste wird in der Variablen size gepseichert.
In unserer Implementierung erzeugen wir uns beim erstellen einer Liste die beiden Objekte head und tail. In diese Objekte werden keine Elemente gespeichert. Wir haben dann zwar aus Speichersicht zwei Objekte "mehr" als wir benötigen würden, aber mit dieser Implementierung können wir uns zur Laufzeit ein paar "Sonderbehandlungen" und Abfragen (z.B. ist dieses Element grad das Erste?) sparen. Bei sehr langen Listen haben wir so einen Vorteil in der Geschwindigkeit und der zusätzliche Speicherbedarf für die beiden Objekte ist sowohl statisch als auch nicht weiter gewichtig. Man kann aber auch in head und tail Daten speichern.
Die Methoden add(int pos, T Element) und add(T Element) sind eigentlich redundant. Bei add ohne Positionsangabe soll das neue Element quasi an der Stelle "size" eingefügt werden, also kann man in dieser Methode einfach die Methode mit Positionsangabe aufrufen, Übergabeparameter sind dann die aktuelle Größe der Liste und das Element.
So kann man Coderedundanz vermeiden. Allerdings haben wir dadurch einige Abfragen, die dann "unnötig" durchlaufen werden. Hier müsste man dann entscheiden, ob man möglichst wenig Redundanzen im Code wünscht oder den Laufzeitvorsprung möchte. Wir haben uns für die Geschwindigkeit entschieden, haben die Alternative aber als Kommentar im Code belassen.
Ähnlich verhält es sich bei den remove Methoden, hier haben wir den Redundanten Teil in eine eigene Methode extrahiert, um Code zu sparen.
Um unsere Implementierung zu realisieren haben wir den Code fortlaufend mit JUnit Tests überprüft, ob die Erwartungen eintreffen und um zu bemerken, wann wir ggf. etwas bereits funktionierendes "kaputt" gespielt haben. Weiterhin ist die Testklasse als Abschlusstest geeignet.

Fehlerbehandlung:

- void add(int pos, E element) throws IndexOutOfBoundsException;
  Falls eine Position angegeben wird, an der eingefügt werden soll, die es nicht gibt.

- E get(int pos) throws IndexOutOfBoundsException;
  Falls eine Position angegeben wird, die es nicht gibt (z.B. größer Listengröße).

- void remove(int pos) throws IndexOutOfBoundsException;
  Falls ein Element an einer Stelle entfernt werden soll, die es nicht gibt in der Liste.

- void remove(E element) throws Exception;
  Wenn es das zu Löschende Element nicht gibt, wird NoSuchElementException geworfen.

Martin Slowikowski
Matrikelnummer: 199 91 66

Jan-Tristan Rudat
Matrikelnummer: 200 78 52

Elementare Java-Operationen:

- statische
  werden bei jedem Funktionsaufruf durchgeführt
- dynamische
  werden nicht immer durchgeführt, hängt z.B. von der Listengröße ab.

Methode	statische Operationen		dyn. Operationen	
void add(int pos, T element)	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	4 1 2 5 2	Vergleiche: Knoten erstellen: - arithmetische: Pointer: Initialisierungen/Zuweisung:	1 - 1 1 -
void add(T element)	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	- 1 1 4 -	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	- - - - -
T get(int pos)	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	4 - 1 1 2	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	1 - 1 1 -
void remove(int pos)	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	3 - 3 3 2	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	1 - 1 1 -
void remove(T element)	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	2 - - - 2	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	2 - 1 2 1
boolean isEmpty()	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	1 - - - -	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	- - - - -
int size()	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	- - - - -	Vergleiche: Knoten erstellen: arithmetische: Pointer: Initialisierungen/Zuweisung:	- - - - -

Martin Slowikowski
Matrikelnummer: 199 91 66

Jan-Tristan Rudat
Matrikelnummer: 200 78 52