Eine verkettete Liste mit Java

Definition: Verkettete Liste

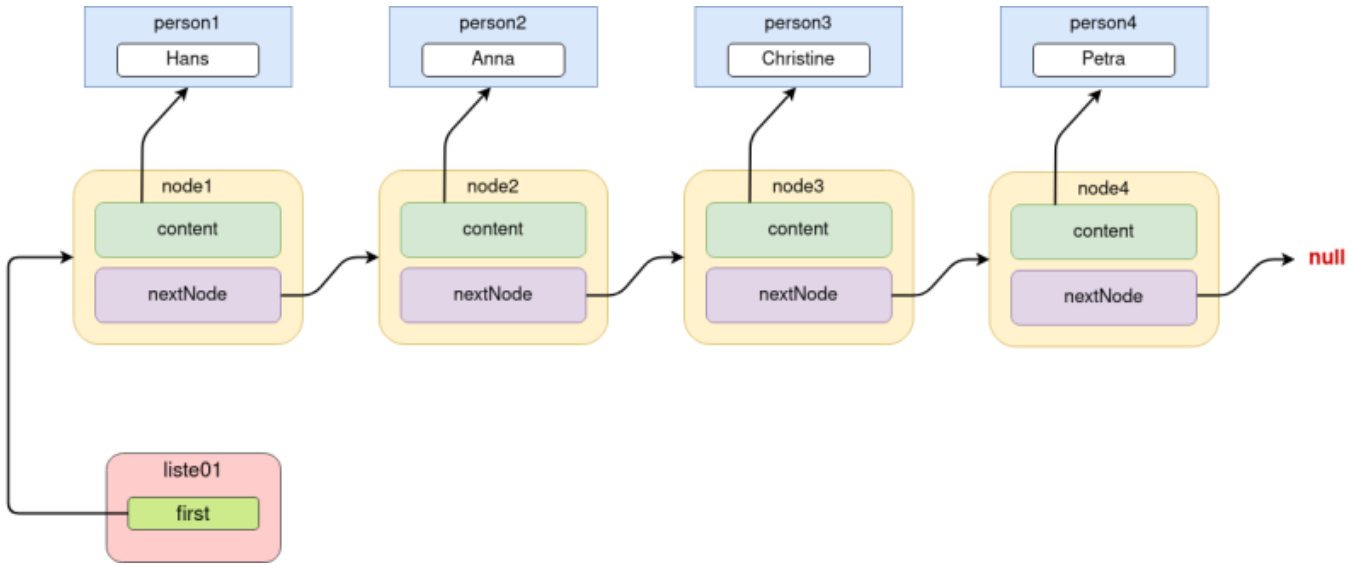
Eine (verkettete) **List** ist ein linearer abstrakter Datentyp mit den folgenden Methoden:

- Konstruktor `List()` – erzeugt eine leere Liste.
- `isEmpty(): boolean` – gibt `true` zurück, wenn die Liste kein Element enthält, sonst wird `false` zurückgegeben.
- `length(): int` – gibt die Anzahl der in der Liste enthaltenen Werte zurück. Eine leere Liste hat die Länge 0. Aufrufe von `append` und `insertAt` erhöhen die Anzahl um 1.
- `getValueAtN(n: int): T` – gibt den Wert an der Position n in der Liste zurück. n muss dabei mindestens 0 und höchstens `length()-1` sein. Falls dies nicht zutrifft, wird null zurückgegeben.
- `append(val: T)` – fügt einen Wert am Ende der Liste ein. Der eingefügte Wert befindet sich nach dem Aufruf an der Stelle `length()-1`.
- `insertAt(index: int, val: T)` – fügt einen Wert an der Position `index` ein. Jeder Wert, der sich hinter der Einfügestelle befindet, befindet sich nach dem Aufruf eine Position weiter hinten. `index` muss mindestens 0 und höchstens `length()` sein. Falls dies nicht zutrifft, wird die Liste nicht verändert. Der eingefügte Wert befindet sich nach dem Aufruf an der Stelle `index`.
- `hasValue(val: T): boolean` – gibt `true` zurück, wenn die Liste einen Wert enthält, der gleich (im Sinne von `equals`) dem Wert val ist, sonst wird `false` zurückgegeben.
- `removeAt(index: int)` – entfernt den Wert, der an der Stelle `index` in der Liste steht, aus der Liste. `index` muss mindestens 0 und höchstens `length()-1` sein. Falls dies nicht zutrifft, wird die Liste nicht verändert.

Erarbeitung

Objektdiagramm

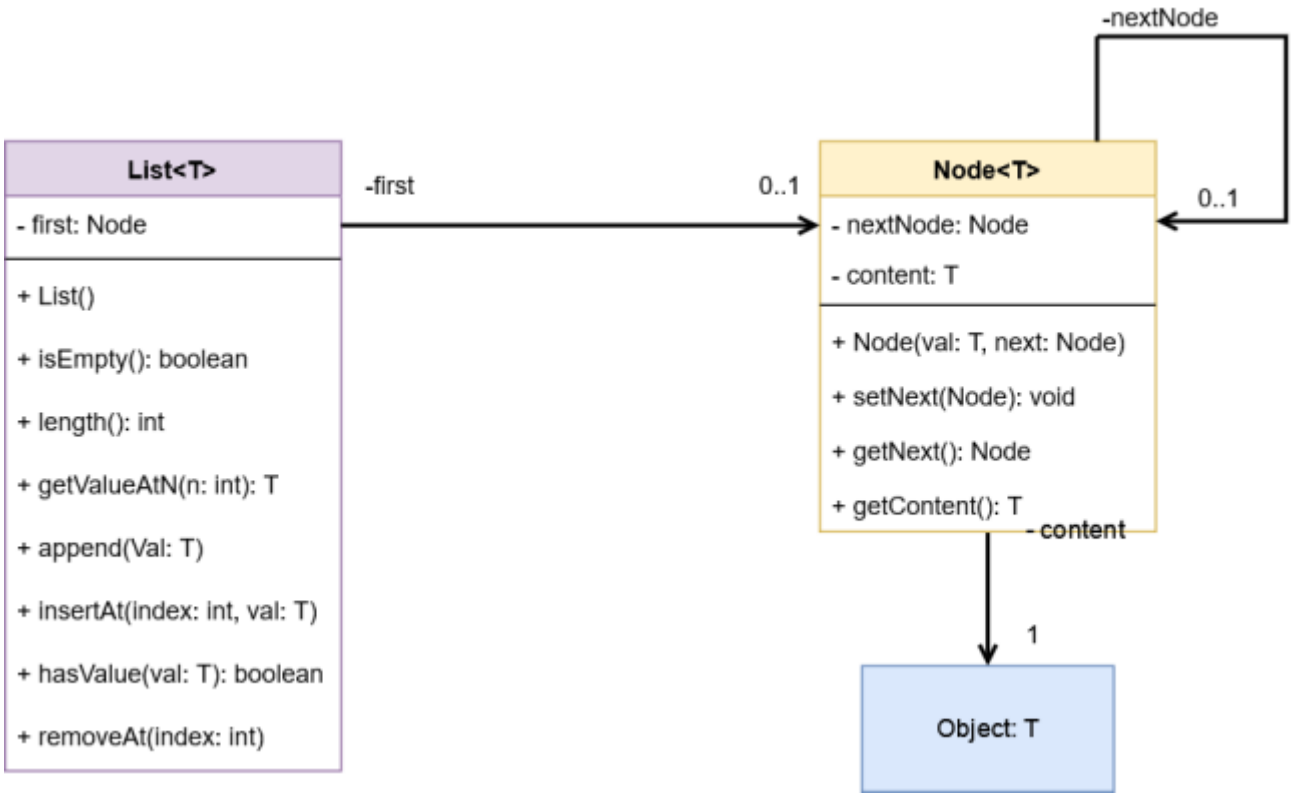
Das **Objektdiagramm** einer Liste sieht folgendermaßen aus.



Die Liste besteht aus verketteten Listenknoten (Objekt: **Node**), jeder Knoten hat (mindestens) zwei Attribute: Eine Referenz auf die mit dem Knoten gespeicherten Daten (`content`) und eine Referenz auf den nächsten Listenknoten (`nextNode`). Das Listenobjekt `liste01` mit seinen Methoden dient der Verwaltung der Knoten und damit der gespeicherten Daten.

Die Listenklasse muss mindestens über ein Attribut `first` verfügen, welches die Referenz auf das erste Knotenobjekt enthält, so dass ausgehend vom ersten Knoten jeder Listenknoten iterativ für weitere Operationen erreichbar ist.

Implementationsdiagramm



Hinweise:

- Liste und Knoten werden als generische Klassen implementiert und mit dem Typ-Parameter T parametrisiert, so dass man beliebige Java Objekte in der Liste verwalten kann.
- Es gibt andere (komfortablere) Möglichkeiten Listen zu implementieren, wir beschränken uns zunächst auf das wesentliche und entwickeln das dann weiter.

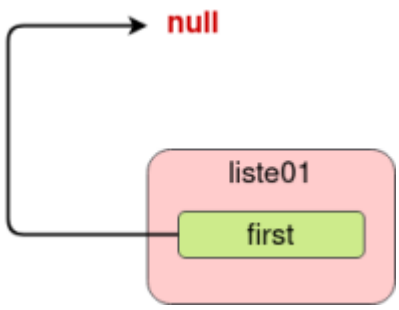
Implementation

Arbeite mit der BlueJ Vorlage von https://codeberg.org/qg-info-unterricht/verkettete-liste-java und bearbeite folgenden Aufgaben, um die Liste gemäß des obigen Implementationsdiagramms zu programmieren.

(A1) Die Klasse ''Node''

Die Klasse `Node` ist in der Vorlage bereits vollständig implementiert. Betrachte den Quellcode, vergleiche ihn mit dem oberen Klassendiagramm und mache dich mit Bedeutung und Funktionsweise der Methoden vertraut. Überprüfe besonders die Typen der Parameter und Rückgabewerte hinsichtlich der Parametrisierung der Klasse mit dem Typparameter T.

(A2) Konstruktor und die Methode ''isEmpty()''



Das Bild zeigt das Objektdiagramm einer leeren Liste. Füge der Klasse `List` einen Konstruktor hinzu, der eine solches Listenobjekt erzeugt. Teste den Konstruktor, indem du ein Listenobjekt erzeugst und mit dem Objektinspektor untersuchst, ob er eine leere Liste wie gewünscht erzeugt.

Hilfe

Beantworte die folgenden Fragen und mache dir klar, was die Antworten für die Implementation des Konstruktors bedeuten:

- Benötigt der Konstruktor Parameter?
- Welche Attribute hat das Listenobjekt? Wie müssen diese bei einer neuen, **leeren** Liste belegt sein?
- Welche Anweisungen musst du deinem Konstruktor hinzufügen, sodass die Attribute die gewünschten Werte haben?

Implementiere dann die Methode `isEmpty`. Welches Kriterium kannst du verwenden, um festzustellen, dass die Liste leer ist? Vergleiche das Objektdiagramm der leeren Liste mit dem der Liste mit Knoten oben auf der Seite.

Teste die Methode zunächst mit deiner leeren Liste.

Lösungsvorschlag

```
 [...]
 /**
 * Konstruktor
 *
 */
public List() {
    this.first = null;
}

[...]

/**
 * Gibt zurück, ob die Liste leer ist.
 * @return true, wenn die Liste keine Elemente enthält; false sonst
 */
public boolean isEmpty() {
    if (this.first == null ) {
        return true;
    }
    return false;
}
```

Weitere Methoden der Liste

Bearbeite die Arbeitsaufträge der Reihe nach, um deine Listenimplementation zu vervollständigen.

- Anhängen eines neuen Elements
- Länge, Wert auslesen
- Einfügen von Elementen
- Löschen von Elementen
- Suche nach Inhaltselement

Überprüfe deine Implementation, wenn diese vollständig ist durch ausführen der automatisierten Tests, indem du auf der grünen Testklasse `ListTest` mit der rechten Maustaste den Punkt `Alles`

`testen` auswählst.

append.odp	201.5 KiB	20.10.2021 16:01
append.pdf	206.8 KiB	20.10.2021 16:01
liste.odp	144.5 KiB	20.10.2021 15:48
liste.pdf	158.3 KiB	20.10.2021 15:48

<<< Zurück zum Vergleich: Arrays und Listen