Grundlagen der Programmkonstruktion
Übungsblatt 3 (zu lösen bis 27./28./24. Mai 2013)

Auf diesem Übungsblatt sollen Sie die folgende Klasse für verkettete Listen um einige Methoden erweitern:

```
public class MyList<A> {
    private Node<A> head = null;

    public boolean isEmpty() {
        return head == null;
    }

    private class Node<A> {
        private A val;
        private Node<A> next = null;

        Node(A v, Node<A> l) {
            val = v;
            next = l;
        }
    }
}
```

Neben den verlangten Methoden können Sie der Klasse `MyList` und auch der inneren Klasse `Node` beliebige weitere Methoden hinzufügen. Achten Sie darauf, dass die Methoden auch dann funktionieren, wenn die Liste leer ist. Verwenden Sie keine anderen Container-Klassen in Ihren Methoden.

1 Methoden in `MyList` hinzufügen

1.1 Methode `int size()` (0.3 Punkte)

Erstellen Sie die Methode `int size()`. Diese soll zurückliefern, wieviele Elemente die Liste enthält.

1.2 Methode `A get(int index)` (0.5 Punkte)

Erstellen Sie die Methode `A get(int index)`. Diese soll das Element mit dem Index `index` in der Liste zurückliefern, wobei das Element, auf das `head` zeigt, den Index 0 hat, das nächste Element den Index 1 usw.

Für ungültige Indices überlegen Sie sich eine geeignete Fehlerbehandlung und begründen Sie sie.

1.3 Vom Anfang der Liste (0.8 Punkt)

Fügen Sie der Klasse `MyList` die Methoden `void addFirst(A n)` und `A removeFirst()` hinzu. Die Methode `void addFirst(A n)` fügt dabei das Element `n` am Anfang der Liste hinzu (das heißt, ein Aufruf von `list.get(0)` liefert dieses Element zurück). Die Methode `A removeFirst()` entfernt das erste Element von der Liste und liefert seinen Inhalt zurück.

1.4 Vom Ende der Liste (1 Punkt)

Fügen Sie der Klasse `MyList` die Methoden `void addLast(A n)` und `A removeLast()` hinzu. Die Methode `void addLast(A n)` fügt dabei das Element `n` am Ende der Liste hinzu (das heißt, ein Aufruf von `list.get(list.size() - 1)` liefert dieses Element zurück). Die Methode `A removeLast()` entfernt das letzte Element von der Liste und liefert seinen Inhalt zurück. Sie können beliebige Methoden der Klasse `MyList` und auch der inneren Klasse `Node` hinzufügen.

2 Queue (0.7 Punkte)

Eine Queue ist eine first-in-first-out-Datenstruktur. In dieser gibt es eine Methoden `void enqueue(A n)` die das Element `n` der Queue hinzufügt (vergleichbar mit `void push(A n)` beim Stack), und `A dequeue()` die ein Element von der Queue entfernt (vergleichbar mit `A pop()`. Im Gegensatz zum Stack soll dabei nicht das zuletzt hinzugefügte Element von der Queue entfernt werden, sondern das Element, das bereits am längsten in der Queue ist.

Erstellen Sie das Interface `QueueI`, das eine solche Queue repräsentiert und implementieren Sie es in `MyList`. Sie dürfen dabei auf alle Methoden aus der vorigen Aufgabe zugreifen.

3 Liste mit Comparable (0.4 Punkte)

Erstellen Sie eine Klasse `MyCompList`, die MyList erweitert. Dabei soll eine Liste vom Typ `MyCompList` nur solche Elemente enthalten können, die das Comparable-Interface implementieren.

Erstellen Sie die Methode `A minimum()`. Diese Methode soll das kleinste Element aus der Liste zurück liefern. Überlegen Sie sich, wie Sie den Fall einer leeren Liste behandeln können.

3.1 Comparator (0.5 Punkte)

Erstellen Sie die Methode `A minimum(Comparator<A> comp)`. In dieser können Sie einen Vergleichsoperator (= Comparator, siehe `http://docs.oracle.com/javase/6/docs/api/java/util/Comparator.html`) angeben. Erstellen Sie einen Comparator comp, sodass der Aufruf `list.minimum(comp)` das größte Element der Liste zurückliefert.