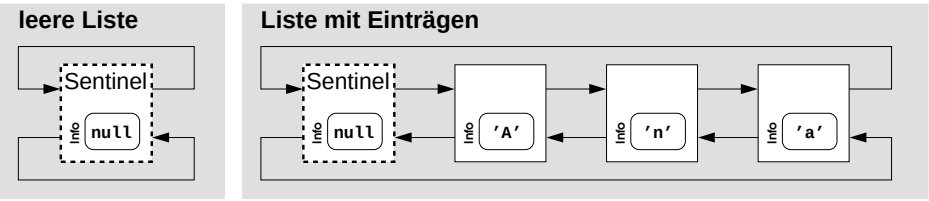
Aufgabe 9.1 (Ü) Doppelt verkettete String-Listen

In dieser Übung soll eine *doppelt verkettete* Liste realisiert werden. In einer solchen Liste wird in jedem Listen-Eintrag (Klasse `Entry`) eine Referenz auf den Vorgänger, eine Referenz auf den Nachfolger sowie eine Referenz auf die eigentliche Information gespeichert. Wir implementieren die doppelt verkettete Liste zyklisch. Dabei gibt es einen ausgezeichneten Listen-Eintrag, der keine Information speichert (das sogenannte *Sentinel-Element*) und sowohl vor dem ersten als auch hinter dem letzten eigentlichen Listen-Eintrag steht (siehe Abbildung). Der erste (bzw. letzte) eigentliche Listen-Eintrag ist dann der Nachfolger (bzw. Vorgänger) des Sentinel-Elementes. Insbesondere wird die leere Liste wie in der nachfolgenden Abbildung repräsentiert.



Implementieren Sie die Klassen `Entry` und `LinkedList`. Ein Objekt der Klasse `LinkedList` soll eine zyklisch doppelt verkettete Liste repräsentieren und muss dementsprechend eine Referenz auf das Sentinel-Element halten. In der Klasse `LinkedList` soll Folgendes definiert sein:

a) Ein Konstruktor zum Erzeugen von leeren Listen.
b) Eine Methode `void append(String s)`, die den String `s` am Ende der Liste anhängt.
c) Eine Methode `String toString()`, die die Liste von Strings ausgibt.
d) `int size()`, die die Länge der Liste zurückliefert;
e) Eine Methode `boolean contains(String s)`, welche darüber Auskunft gibt, ob eine Liste einen String, der `s` gleicht, enthält.

Lösungsvorschlag 9.1

Achtung: Diese Klasse am Besten Schritt für Schritt wachsen lassen, nicht gleich von Anfang an z.B. das Size-Attribut mitführen. Ruhig zeigen, dass man Methoden auch mal im Nachhinein nochmal anpassen muß. Beim contains() darauf hinarbeiten, dass man die null-Pointer Problematik anspricht. Entweder man geht davon aus, dass die Liste nie um null-Strings erweitert werden konnte (dann muß append() angepasst werden), oder man passt hier beim .equals() auf!

```
public class Entry {
        String element;
        Entry next;
        Entry previous;
```

```
        Entry(String element, Entry next, Entry previous) {
                this.element = element;
                this.next = next;
                this.previous = previous;
        }

        public String getElement() {
                return element;
        }

        public String toString() {
                return "[Entry : " + element + "]";
        }
}

public class LinkedList {
    private Entry sentinel;
    public LinkedList() {
        size = 0; // Achtung - erst ab Teilaufgabe d hinzugefuegt
        sentinel = new Entry(null, null, null);
        sentinel.previous = sentinel.next = sentinel;
    }

    public void append(String s){
        Entry insertBefore = sentinel;
        if (s!=null){ // Konsistenzbedingung: Die Liste enthaelt keine
            null ausser im Sentinel
            Entry newEntry = new Entry(s, insertBefore, insertBefore.
                previous);
            newEntry.previous.next = newEntry;
            newEntry.next.previous = newEntry;
            size++; // Achtung - erst ab Teilaufgabe d hinzugefuegt
        }
    }
    public String toString() {
        String retVal = "[";
        for (Entry e = sentinel.next; e != sentinel; e = e.next) {
            retVal = retVal + e.getElement();
            if (e.next != sentinel) // Man ist nicht bem letzten
                Element, also "," ausgeben
                {
                    retVal = retVal + ",";
                }
        }
        return retVal + "]";
    }

    private int size;
    public int size(){
        return size;
    }
    public boolean contains(String s){
        for (Entry e = sentinel.next; e != sentinel; e = e.next) {
            if (e.getElement().equals(s))
                return true;
        }
        return false;
```

```
    }


}
```

Aufgabe 9.2 (H) Doppelt verkettete String-Listen II

Erweitern Sie Ihre Implementierung aus Aufgabe 9.1 um folgende Methoden:

a) `void addFirst(String s)`, die den String `s` am Anfang der Liste einfügt;
b) `String first()` und `String last()`, die, falls die Liste nicht leer ist, den ersten bzw. den letzten echten Listeneintrag zurückliefern. Ist die Liste leer, soll `null` zurückgeliefert werden;
c) Eine private Methode `void remove(Entry e)`, die genau das Element `e` aus der Liste lÃ¶scht, sofern es nicht das `sentinel` ist.
d) `void removeFirst()` und `void removeLast()`, die den ersten bzw. letzten Eintrag löschen, sowie `void remove(String s)`, die einen String, der `s` gleicht, aus der Liste löschen, wenn vorhanden.

Vermeiden Sie redundanten Code durch sinnvolle Verwendung bereits implementierter Methoden!

Lösungsvorschlag 9.2

```
public class LinkedListH {
    private Entry sentinel;
    private int size;


    public LinkedListH() {
        size = 0;
        sentinel = new Entry(null, null, null);
        sentinel.previous = sentinel.next = sentinel;
    }

    public void append(String s){
        if (s!=null){
            Entry newEntry = new Entry(o, sentinel, sentinel.previous);
            newEntry.previous.next = newEntry;
            newEntry.next.previous = newEntry;
            size++;
        }
    }
    public String toString() {
        String retVal = "[";
        for (Entry e = sentinel.next; e != sentinel; e = e.next) {
            retVal = retVal + e.getElement();
            if (e.next != sentinel) // Man ist nicht bem letzten
               Element, also "," ausgeben
                {
                    retVal = retVal + ",";
                }
        }
        return retVal + "]";
    }
```

```
public int size(){
    return size;
}
public boolean contains(String s){
    for (Entry e = sentinel.next; e != sentinel; e = e.next) {
        if (e.getElement().equals(s))
            return true;
    }
    return false;
}



//Teilaufabe a)
private void addBefore(String o, Entry e) {
    if (o!=null){
        Entry newEntry = new Entry(o, e, e.previous);
        newEntry.previous.next = newEntry;
        newEntry.next.previous = newEntry;
        size++;
    }
}
public void addFirst(String o) {
    addBefore(o, sentinel.next);
}


//Teilaufabe b)
public String first() {
    return sentinel.next.element;
}
public String last(){
    return sentinel.previous.element;
}


//Teilaufgabe c)
private void remove(Entry e) {
    if (e != sentinel){
        e.previous.next = e.next;
        e.next.previous = e.previous;
        size--;
    }
}

//Teilaufgabe d)
public void removeFirst(){
    if(first()!=null) remove(sentinel.next);
}
public void removeLast(){
    if(last()!=null) remove(sentinel.previous);
}
public void remove(String s){
    Entry anchor = sentinel.next;
    while (anchor!=sentinel){
        if (anchor.element.equals(s)){
            remove(anchor);
```

```
                }
                anchor=anchor.next;
            }
        }



}
```