# INLOOP Softwaretechnologie, Linked List

Henry Haustein

## vollständiger Code

Datei `ListElement.java`

```java
public class ListElement {
  private String content;
  private ListElement next;

  public ListElement(String con) {
    if(con == null || con.equals("")) {
      throw new IllegalArgumentException("content must not be null");
    }
    else {
      this.content = con;
      this.next = null;
    }
  }

  public String getContent() {
    return content;
  }

  public ListElement getNext() {
    return next;
  }

  public void setContent(String con) {
    if(con == null || con.equals("")) {
      throw new IllegalArgumentException("content must not be null");
    }
    else {
      content = con;
    }
  }

  public void setNext(ListElement nx) {
    next = nx;
  }
}
```

Datei `List.java`

```java
public class List {
  private ListElement head;

  public List() {
    this.head = null;
  }

  public void append(String con) {
    ListElement elem = new ListElement(con);

    // 0 Elements
    if(head == null) {
      head = elem;
    }
    else {
      // 1 Element = head
      if(head.getNext() == null) {
        head.setNext(elem);
      }
      // more than 1 Element
      else {
        ListElement current = head;
        while(current != null) {
          if(current.getNext() == null) {
            System.out.println("Found end at " + current.getContent());
            current.setNext(elem);
            break;
          }
          else {
            System.out.println("Go to next element of " + current.getContent());
            current = current.getNext();
          }

        }
      }
    }
  }

  public String remove(String toRemove) {
    if(toRemove == null || toRemove.equals("")) {
      throw new IllegalArgumentException("content must not be null");
    }

    // 0 Elements
    if(head == null) {
      return null;
    }
    else {
      // 1 Element
      if(head.getNext() == null) {
        if(head.getContent().equals(toRemove)) {
```

```
52            String content = head.getContent();
53            head = null;
54            return content;
55          }
56          else {
57            return null;
58          }
59        }
60        // more than 1 Element
61        else {
62          // must head be removed ?
63          if(head.getContent().equals(toRemove)) {
64            String content = head.getContent();
65            head = head.getNext();
66            return content;
67          }
68          else {
69            ListElement current = head.getNext();
70            ListElement prev = head;
71            while(true) {
72              if(current.getContent().equals(toRemove)) {
73                prev.setNext(current.getNext());
74                return current.getContent();
75              }
76              else {
77                if(current.getNext() == null) {
78                  return null;
79                }
80                else {
81                  prev = current;
82                  current = current.getNext();
83                }
84              }
85            }
86          }
87        }
88      }
89    }
90
91    public void printList() {
92      if(head == null) {
93        System.out.println("List is empty");
94      }
95      else {
96        ListElement current = head;
97        while(current != null) {
98          System.out.println(current.getContent());
99          current = current.getNext();
100       }
101     }
102   }
```

```
103
104 }
```

## Erklärung

Auch hier ist die Klasse `ListElement` recht klar, der Operator `||` steht für "oder"[1].

Die Idee hinter der `append()`-Funktion ist, dass es für das Anfügen an eine Liste eigentlich nur 2 Fälle gibt:

- Die Liste ist leer, dass besteht die neue Liste nur aus dem angefügten Objekt.
- Die Liste ist nicht leer, wir müssen also das letzte Element finden, und hängen an dieses Element das anzufügende Element dran.

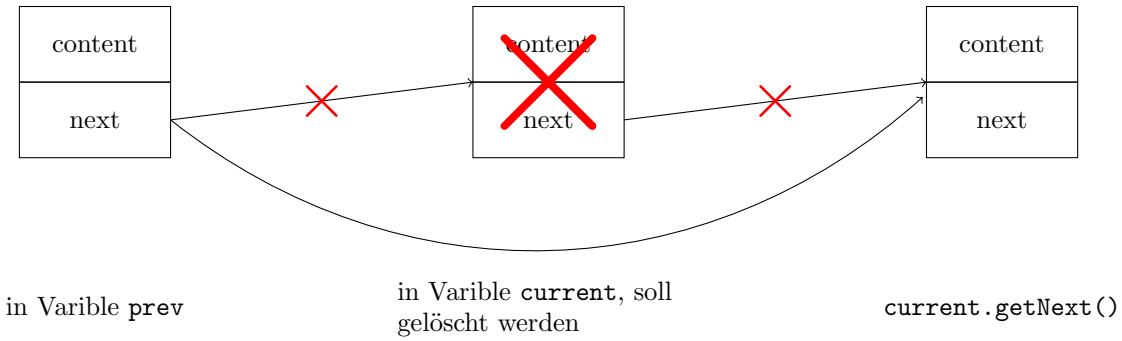
Ich habe hier tatsächlich noch mal eine Unterscheidung gemacht, ob die Liste nur aus einem Element besteht, oder ob es noch mehr Elemente gibt. Eigentlich sollte diese Unterscheidung nicht notwendig sein und ich denke auch, dass mein Code ohne Probleme funktionieren sollte, wenn man die Zeilen 16 bis 21 aus meinem Quelltext entfernt, aber der Code funktioniert. Wissen wir nun, dass die Liste mehr als ein Element enthält, so setzen wir unser aktuell betrachtetes Element auf `head`. Wir gehen nun solange zum Nachfolger des aktuell betrachteten Elements, bis wir feststellen, dass das aktuelle Element keinen Nachfolger hat, also `current.getNext() == null`. Das verhindert auch, dass die Bedingung in der while-Schleife jemals falsch sein wird, ich hätte auch also auch `while(true)` statt `while(current != null)` schreiben können. Haben wir nun das letzte Element gefunden, so können wir an dieses den Nachfolger dranhängen und brechen die while-Schleife ab.

Die `remove()`-Funktion ist noch komplizierter. Grundsätzliche Idee ist auch hier, dass wir überprüfen, ob der Nachfolger des aktuell betrachteten Elements gelöscht werden muss. Wenn dem so ist, müssen wir den Nachfolger des aktuell betrachteten Elements auf `null` setzen. Das impliziert, dass diese Vorgehensweise nur dann funktioniert, wenn in der Liste mindestens 2 Elemente sind, wir müssen also die Fälle mit 0 Elementen in der Liste und mit einem Element in der Liste separat bearbeiten.

- Wenn es keine Elemente in der Liste gibt, so kann auch nichts entfernt werden, und wir geben `null` zurück.
- Ist nur ein Element in der Liste, so müssen wir überprüfen, ob dieses gelöscht werden muss. Ist dem so, so muss die ganze Liste gelöscht werden (`head = null;`). Vorher haben ich noch den Inhalt von `head` abgespeichert, so dass ich ihn nach dem Löschen der Liste zurückgeben kann.

Ein weiterer Spezialfall ist, dass die Liste mehr als 1 Element enthält, aber das erste Element gelöscht werden muss. In diesem Fall muss man die `head`-Variable einfach auf das zweite Listenelement setzen. Sind nun diese ganzen Spezialfälle nicht eingetreten, so setzen wir die Variable `current` auf das zweite Element der Liste und die Variable `prev` (wie *previous*, also Vorgänger) auf das erste Element der Liste. Die Variablenbezeichnung mag vielleicht etwas unintuitiv sein, aber das Element, was in `current` drin steht, soll auf Löschbarkeit überprüft werden, und in dem Fall, dass dieses gelöscht werden muss, darf der Vorgänger dieses Elements nicht mehr auf `current` zeigen, sondern muss auf `current.getNext()` zeigen.

[1] Der Operator `&&` steht für "and".




Sind wir am Ende der Liste angekommen (`current.getNext() == null`), ohne etwas zu finden, so geben wir `null` zurück. Gibt es hingegen noch einen Nachfolger, so updaten wir `prev` und `current`.

Zu Testzwecken habe ich mir noch eine Funktion `printList()` geschrieben, die sich durch die Liste iteriert und einfach den Inhalt der einzelnen Elemente ausgibt.