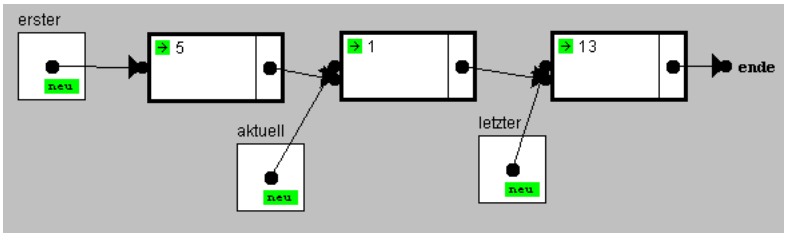
Die Klasse List in Java

Erweiterungen „ListErw“ + „ListInt“

Die generische Klasse ListErw <ContentType>, abgeleitet von List <Content Type>

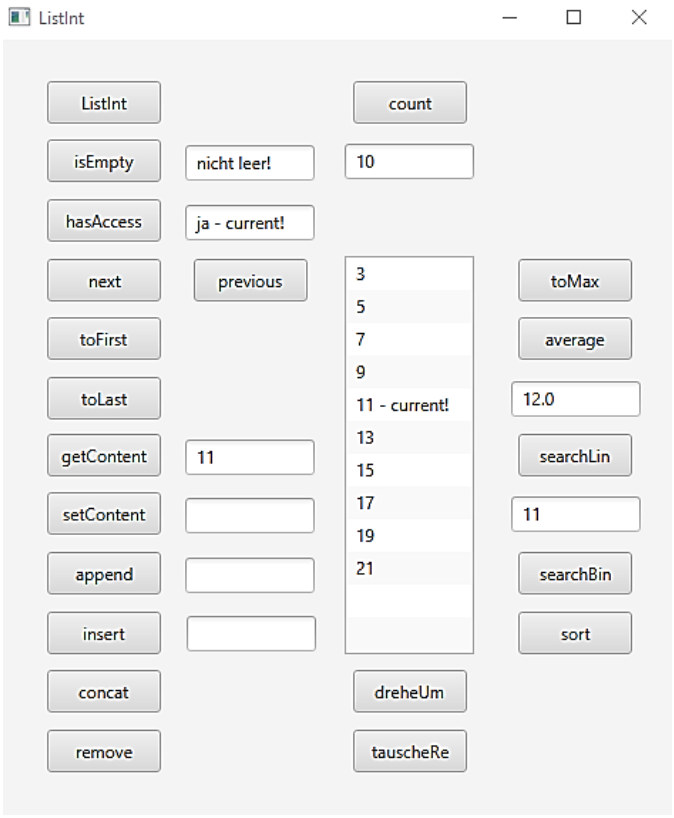
```
public class ListErw <ContentType> extends List <ContentType> {
   public int count() {
      int i=0;
      if (!isEmpty()) {
         ListNode help = current; //zwischenspeichern
         toFirst(); while (hasAccess()) { next(); i++; }
         current = help; //zurücksetzen
      }
      return i;
   }

   public void dreheUm() {
      if (first != last) {
         ListNode help1 = last;
         while (help1 != first) {
            ListNode help2 = help1;
            help1 = getPrevious(help1); //help1 vor help2
            help2.setNextNode(help1);
         }
         help1.setNextNode(null); first = last; last = help1;
      }
   }

   public void tauscheRechts() {
      if (hasAccess() && !(current == last)) {
         ListNode help = current.getNextNode(); //current vor help
         current.setNextNode(help.getNextNode());
         help.setNextNode(current); //current hinter help, help aber ohne Vorgänger!
         if (current == first) {
            first = help; //help wird verbunden (current bleibt dahinter)
         }
         else {
            current = getPrevious(current); //current geht zum alten Vorgänger
            current.setNextNode(help); //help wird verbunden (current vor help)
            current = help.getNextNode(); //current wieder hinter help
         }
         if (help == last) last = last.getNextNode();
      }
   }
}
```



Die bekannte, „normale“ Liste wird mittels **Vererbung** um schöne und nützliche Methoden erweitert, die aufgrund der Namensgebung im Prinzip selbsterklärend sind.

Die generische Klasse ListInt, abgeleitet von ListErw <Content Type>

ListErw wird nochmals **vererbt**, um im Speziellen Listen aus ganzen Zahlen schöner und detaillierter analysieren zu können.




```
public class ListInt extends ListErw <Integer> {
   public void toMax() {
      if (!isEmpty()) {
         ListNode help = first; toFirst(); next();
         while (hasAccess()) {
            if (getContent() > help.getContentObject()) {
               help = current; } next();
         } current = help;
      }
   }
   public double average() {
      double erg = 0;
      if (!isEmpty()) {
         ListNode help = current; //zwischenspeichern
         toFirst(); while (hasAccess()) {
           erg = erg + getContent(); next(); }
         erg = erg/count(); current = help; //zurücksetzen
      } return erg;
   }
   public void searchLin(int x) {
      //current geht zu x (1. Auftreten) oder wird null
      toFirst();
      int zaehler=1; //zählt die Anzahl an Durchläufen
      while(hasAccess() && getContent() != x) {
         next(); zaehler ++; } System.out.println(zaehler);
   }
   public void searchBin(int x) {
      //Zahlen müssen in sortierter Form vorliegen!
      int l=1;  int r=count(); int zaehler=0; //zählt die Anzahl an Durchläufen
      while (l<=r) { int m = (l+r)/2; zaehler ++; toFirst(); for (int i=1; i<m; i++) next(); //current zu "m"
         if (getContent() < x) l=m+1; else if (getContent() > x) r=m-1; else l=r+2;
      } if (l==r+1) current = null; System.out.println(zaehler);
   }
   public void sort() { //BubbleSort
      ListNode help = current; //zwischenspeichern
      int zaehler=0; //zählt die Anzahl an Vergleichen
      for (int i=1; i<count(); i++) { toFirst();
         for (int j=0; j<count()-i; j++) {
            int nachbar = current.getNextNode().getContentObject();
            if (getContent() > nachbar) tauscheRechts(); else next(); zaehler ++;
         }
      } current = help; //zurücksetzen
      System.out.println(zaehler);
   }
}
```