METHODE	PSEUDOCODE	IMPLEMENTIERUNG (JAVA-CODE)
insert	**WENN** der Knoteninhalt nicht leer ist **WENN** es einen aktuellen Knoten gibt **DANN** erzeuge neuen Knoten **WENN** der current-Knoten nicht der first-Knoten ist **DANN** gib den Vorgänger (previous-Knoten) vom current-Knoten an Current-Knoten wird Nachfolger des neuen Knotens Neuer Knoten wird Nachfolger des previous-Knotens **SONST** mache den first-Knoten zum Nachfolger des neuen Knotens ernenne den neuen Knoten zum first-Knoten **SONST** **WENN** die Liste leer ist **DANN** erzeuge neuen Knoten mache den neuen Knoten zum first-Knoten mache den neuen Knoten zum last- Knoten	public void insert(ContentType pContent) { if (pContent != null) *{//Nichts tun, wenn kein Inhalt* if (this.hasAccess()) {*//Fall: Es gibt ein aktuelles Element* // Neuen Knoten erstellen. ListNode newNode = new ListNode(pContent); if (current != first) {*//Fall: Nicht an 1. Stelle einfügen.* ListNode previous = this.getPrevious(current); newNode.setNextNode(previous.getNextNode()); previous.setNextNode(newNode); } else {*//Fall: An 1. Stelle einfügen.* newNode.setNextNode(first); first = newNode; } } else {*//Fall: Es gibt kein aktuelles Element.* if (this.isEmpty()) {*//Fall: In leere Liste einfuegen.* *// Neuen Knoten erstellen.* ListNode newNode = new ListNode(pContent); first = newNode; last = newNode; } } } }
append	**WENN** der Knoten nicht leer ist **WENN** die Liste leer ist **DANN** füge neuen Knoten ein, indem du insert aufrufst **SONST** Erzeuge neuen Knoten Der neue Knoten wird Nachfolger des aktuell letzten Knotens Der neue Knoten wird zum letzten Knoten	*Falls pContent gleich null ist, geschieht nichts. Ansonsten wird ein neues Objekt pContent am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt pContent in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (hasAccess() == false).* public void append(ContentType pContent) { if (pContent != null) {*//Nichts tun, wenn es keine Inhalt gibt.* if (this.isEmpty()) {*//Fall: An leere Liste anfügen* this.insert(pContent); } else {*// Fall: An nicht-leere Liste anfügen* //Neuen Knoten erstellen. ListNode newNode = new ListNode(pContent); last.setNextNode(newNode); last = newNode; //Letzten Knoten aktualisieren } } }

remove	**WENN** es kein aktuelles Element gibt oder die Liste leer ist **WENN** der current-Knoten gleichzeitig der first-Knoten ist **DANN** ernenne den zweiten Knoten zum first-Knoten **SONST** Mache den Vorgänger vom current-Knoten zum previous-Knoten **WENN** der current-Knoten der last-Knoten ist Ernenne den previous-Knoten zum last-Knoten	*Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (hasAccess() == false), geschieht nichts. Falls es ein aktuelles Objekt gibt (hasAccess() == true), wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr.* `public void remove() {` `  // Nichts tun, wenn es kein aktuelle Element gibt oder die Liste leer ist.` `  if (this.hasAccess() && !this.isEmpty()) {` `  if (current == first) {` `  first = first.getNextNode();` `  } else {` `  ListNode previous = this.getPrevious(current);` `  if (current == last) {` `  last = previous;` `  }` `  previous.setNextNode(current.getNextNode());` `  }` `  ListNode temp = current.getNextNode();` `  current.setContentObject(null);` `  current.setNextNode(null);` `  current = temp;` `  //Beim Löschen des letzten Elements last auf null setzen` `  if (this.isEmpty()) {` `  last = null;` `  }` `  }` `  }`