

Evgen959 / Advanced_Backend Public

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#)

Advanced_Backend / Lesen 037 / code / d06_3_2 / src / MyLikedList.java

...



Evgen959 newLes37

078446a · 7 minutes ago



185 lines (159 loc) · 4.07 KB

Code

Blame

Raw



```
1  import java.util.Iterator;
2
3  public class MyLikedList<E> implements MyList<E>{
4
5      private Node<E> head = null;
6      private Node<E> tail = null;
7      private int size = 0;
8
9      @Override
10     public boolean add(E element) {
11         Node<E> node = new Node<>(tail, null, element);
12         size++;
13         if (tail!=null){
14             tail.setNext(node);
15         }
16         if (head==null){
17             head = node;
18         }
19         tail = node;
20         return true;
21     }
22
23     public boolean isEmpty(){
24         return head==null;
25     }
26
27     // 0...size-1
28     @Override
29     public boolean add(int index, E element) {
30         if (index>=size){
31             return add(element);
32         }
33         Node<E> node = new Node<>(null,null,element);
34         Node<E> next = getNode(index);
35         if(next == null||index<=0){ //добавляем ноду в 0 индекс
36             next=head;
37             head=node;
```

```
38         }
39         Node<E> prev = next.getPrev();
40         next.setPrev(node);
41         node.setNext(next);
42         node.setPrev(prev);
43         if (prev!=null){
44             prev.setNext(node);
45         }
46         size++;
47         return false;
48     }
49
50     @Override
51     public E get(int index) {
52         Node<E> node = getNode(index);
53         return (node!=null)?node.getValue():null;
54     }
55
56     private Node<E> getNode(int index){
57         if (index>=size || index<0 || head==null){
58             return null;
59         }
60         int counter = 0;
61         Node<E> aktiveNode = head;
62         while (aktiveNode!=null && counter<index){
63             aktiveNode = aktiveNode.getNext();
64             counter++;
65         }
66         return aktiveNode;
67     }
68
69     @Override
70     public int size() {
71         return size;
72     }
73
74     private E remove(Node<E> node){// удаляем ноду
75         if (node==null){
76             return null;
77         }
78         Node<E> prev = node.getPrev();
79         Node<E> next = node.getNext();
80
81         if (prev!=null){
82             prev.setNext(next);
83         } else {
84             head = next;
85         }
86         if (next!=null){
87             next.setPrev(prev);
88         } else {
89             tail = prev;
90         }
91     }
```

```
91         size--;
92         node.setPrev(null);
93         node.setNext(null);
94         E removedValue = node.getValue();
95         return removedValue;
96     }
97
98     @Override
99     public E remove(int index) {
100         Node<E> node = getNode(index); // ищет ноду
101         return remove(node);
102     }
103
104     @Override
105     public E remove() {
106         return remove(tail);
107     }
108
109     @Override
110     public E set(int index, E element) {
111         return null;
112     }
113
114     @Override
115     public String toString() {
116         if (head==null){
117             return "[]";
118         }
119         StringBuilder sb = new StringBuilder();
120         Node<E> currentNode = head;
121         while (currentNode!=null){
122             sb.append(currentNode.getValue()).append(";");
123             currentNode=currentNode.getNext();
124         }
125         sb.setLength(sb.length()-1);
126         return "[" + sb.toString() + ']';
127     }
128
129     @Override
130     public Iterator<E> iterator(){
131         return this.new MyListIteraror();
132     }
133
134     private class MyListIteraror implements Iterator<E>{
135         private Node<E> current = head;
136
137         @Override
138         public boolean hasNext() {
139             return current!=null;
140         }
141
142         @Override
143         public E next() {
```

```
144         E value = current.value;
145         current=current.next;
146         return value;
147     }
148 }
149
150 ✓ public class Node<E> {
151     Node<E> prev;
152     Node<E> next;
153     E value;
154
155 ✓ public Node(Node<E> prev, Node next, E value) {
156     this.prev = prev;
157     this.next = next;
158     this.value = value;
159 }
160
161 public Node<E> getPrev() {
162     return prev;
163 }
164
165 public void setPrev(Node<E> prev) {
166     this.prev = prev;
167 }
168
169 public Node<E> getNext() {
170     return next;
171 }
172
173 public void setNext(Node<E> next) {
174     this.next = next;
175 }
176
177 public E getValue() {
178     return value;
179 }
180
181 public void setValue(E value) {
182     this.value = value;
183 }
184 }
185 }
```