

# Übungsblatt 12

Adrian Schollmeyer

## Aufgabe 1

Listing 1: DoubleLinkedList.java

```
1 package me.adrian.aup12;
2
3 import java.util.NoSuchElementException;
4
5 public class DoubleLinkedList<T> implements Iterable<T>
6 {
7     private ListItem<T> first = null;
8     private ListItem<T> last = null;
9
10    public DoubleLinkedList()
11    {
12    }
13
14    public void addFirst(T prependedValue)
15    {
16        ListItem<T> newFirst = new ListItem<>(prependedValue, null, this.first);
17
18        if (this.first != null)
19            this.first.prev = newFirst;
20
21        this.first = newFirst;
22
23        if (this.last == null)
24            this.last = this.first;
25    }
26
27    public void addLast(T appendedValue)
28    {
29        ListItem<T> newLast = new ListItem<>(appendedValue, this.last, null);
30
31        if (this.last != null)
32            this.last.next = newLast;
33
34        this.last = newLast;
35
36        if (this.first == null)
37            this.first = this.last;
38    }
39
40    public T getFirst()
```

```
41     {
42         this.assertNotEmpty();
43         return this.first.value;
44     }
45
46     public T getLast()
47     {
48         this.assertNotEmpty();
49         return this.last.value;
50     }
51
52     public T removeFirst()
53     {
54         this.assertNotEmpty();
55
56         ListItem<T> currentFirst = this.first;
57         if (this.size() == 1) {
58             this.first = null;
59             this.last = null;
60         } else {
61             this.first = currentFirst.next;
62             this.first.prev = null;
63         }
64
65         return currentFirst.value;
66     }
67
68     public T removeLast()
69     {
70         this.assertNotEmpty();
71
72         ListItem<T> currentLast = this.last;
73         if (this.size() == 1) {
74             this.first = null;
75             this.last = null;
76         } else {
77             this.last = currentLast.prev;
78             this.last.next = null;
79         }
80
81         return currentLast.value;
82     }
83
84     public int size()
85     {
86         if (this.first == null || this.last == null)
87             return 0;
88
89         return this.last.getIndex() + 1;
```

```
90     }
91
92     private void assertNotEmpty()
93     {
94         if (this.size() <= 0)
95             throw new IndexOutOfBoundsException("Empty list");
96     }
97
98     @Override
99     public boolean equals(Object other)
100    {
101        if (this == other)
102            return true;
103
104        if (other instanceof DoubleLinkedList) {
105            DoubleLinkedList nl = (DoubleLinkedList) other;
106
107            if (this.size() != nl.size())
108                return false;
109
110            Iterator otherIterator = (Iterator) nl.iterator();
111
112            for (T thisItem : this) {
113                if (!thisItem.equals(otherIterator.next())) {
114                    return false;
115                }
116            }
117            return true;
118        } else {
119            return false;
120        }
121    }
122
123    @Override
124    public int hashCode()
125    {
126        if (this.size() == 0)
127            return 0;
128
129        return 31 * this.first.recursiveHashCode();
130    }
131
132    @Override
133    public java.util.Iterator iterator()
134    {
135        return new Iterator(this.first);
136    }
137
138    public java.util.Iterator reverseIterator()
```

```
139     {
140         return new ReverseIterator(this.last);
141     }
142
143     public static class Iterator<T> implements java.util.Iterator
144     {
145         private ListItem<T> item;
146
147         private Iterator(ListItem<T> item)
148         {
149             this.item = item;
150         }
151
152         @Override
153         public boolean hasNext()
154         {
155             return this.item != null;
156         }
157
158         @Override
159         public T next()
160         {
161             if (this.item == null)
162                 throw new NoSuchElementException("Trying to access elements past
the last element");
163
164             T value = this.item.value;
165             this.item = this.item.next;
166
167             return value;
168         }
169     }
170
171     public static class ReverseIterator<T> implements java.util.Iterator
172     {
173         private ListItem<T> item;
174
175         private ReverseIterator(ListItem<T> item)
176         {
177             this.item = item;
178         }
179
180         @Override
181         public boolean hasNext()
182         {
183             return this.item != null;
184         }
185
186         @Override
```

```
187     public T next()
188     {
189         if (this.item == null)
190             throw new NoSuchElementException("Trying to access elements
191                         before the first element");
192
193         T value = this.item.value;
194         this.item = this.item.prev;
195
196         return value;
197     }
198
199     private static class ListItem<T>
200     {
201         private T value;
202         private ListItem<T> prev;
203         private ListItem<T> next;
204
205         private ListItem(T initialValue, ListItem<T> prev, ListItem<T> next)
206         {
207             this.value = initialValue;
208             this.prev = prev;
209             this.next = next;
210         }
211
212         public int getIndex()
213         {
214             if (this.isFirstElement())
215                 return 0;
216
217             int thisIndex = this.prev.getIndex() + 1;
218
219             return thisIndex;
220         }
221
222         public boolean isFirstElement()
223         {
224             return this.prev == null;
225         }
226
227         public boolean isLastElement()
228         {
229             return this.next == null;
230         }
231
232         public int recursiveHashCode()
233         {
234             return this.value.hashCode() + 31 * (this.isLastElement() ? 0 : this.
```

```
        next.recursiveHashCode());  
235    }  
236}  
237}
```

## Aufgabe 2

Listing 2: Stack.java

```
1 package me.adrian.aup12;  
2  
3 public class Stack<T>  
4 {  
5     private DoubleLinkedList<T> data = new DoubleLinkedList();  
6  
7     public Stack()  
8     {  
9     }  
10  
11    public void push(T value)  
12    {  
13        data.addFirst(value);  
14    }  
15  
16    public void pop()  
17    {  
18        data.removeFirst();  
19    }  
20  
21    public T top()  
22    {  
23        return data.getFirst();  
24    }  
25  
26    public boolean isEmpty()  
27    {  
28        return this.data.size() == 0;  
29    }  
30  
31    @Override  
32    public boolean equals(Object other)  
33    {  
34        if (this == other)  
35            return true;  
36  
37        if (other instanceof Stack) {  
38            Stack ns = (Stack) other;  
39            return this.data.equals(ns.data);  
40        }  
41    }
```

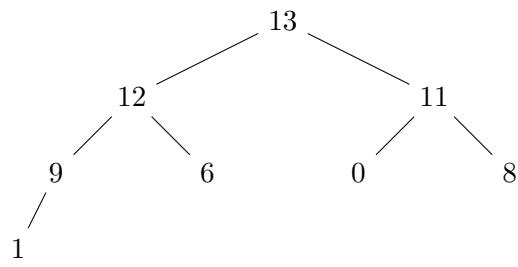
```

42         return false;
43     }
44
45     @Override
46     public int hashCode()
47     {
48         return this.data.hashCode();
49     }
50 }
```

### Aufgabe 3

(a)

- |   |   |
|---|---|
| (1) Heap leer, 9 wird Wurzelknoten<br> 9  | (9) 0 einfügen, wird linkes Kind von 8, keine Vertauschung, da nicht größer als 8<br> 13 9 8 1 6 0              |
| (2) 1 einfügen, wird linkes Kind von 9, keine Vertauschung, da nicht größer als 9<br> 9 1                     | (10) 11 einfügen, wird rechtes Kind von 8<br> 13 9 8 1 6 0 11   |
| (3) 8 einfügen, wird rechtes Kind von 9, keine Vertauschung, da nicht größer als 9<br> 9 1 8                  | (11) 11 mit 8 vertauschen, da $11 > 8$<br> 13 9 11 1 6 0 8  |
| (4) 6 einfügen, wird linkes Kind von 1<br> 9 1 8 6  | (12) 12 einfügen, wird linkes Kind von 1<br> 13 9 11 1 6 0 8 12   |
| (5) 6 mit 1 vertauschen, da $6 > 1$ , danach keine Vertauschung mehr, da $6 \leq 9$<br> 9 6 8 1               | (13) 12 mit 1 vertauschen, da $12 > 1$<br> 13 9 11 12 6 0 8 1   |
| (6) 13 einfügen, wird rechtes Kind von 6<br> 9 6 8 1 13   | (14) 12 mit 9 vertauschen, da $12 > 9$ , danach keine Vertauschung mehr, da $12 \leq 13$<br> 13 12 11 9 6 0 8 1 |
| (7) 13 mit 6 vertauschen, da $13 > 6$<br> 9 13 8 1 6  |   |
| (8) 13 mit 9 vertauschen, da $13 > 9$ , danach keine Vertauschung mehr, da 13 nun Wurzelknoten<br> 13 9 8 1 6 |   |



(b)

- (1) 13 entfernen und 1 an diese Stelle setzen

|1|12|11|9|6|0|8|

- (2) 12 mit 1 vertauschen, da  $12 > 1$  und 12 größtes Kind von 1

|12|1|11|9|6|0|8|

- (3) 9 mit 1 vertauschen, da  $9 > 1$  und 9 größtes Kind von 1

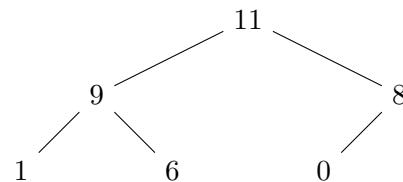
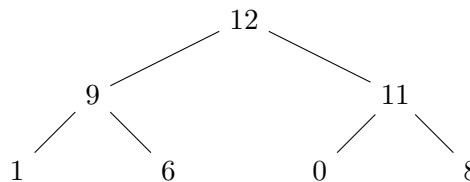
|12|9|11|1|6|0|8|

- (1) 12 entfernen und 8 an diese Stelle setzen

|8|9|11|1|6|0|

- (2) 11 mit 8 vertauschen, da  $11 > 8$  und 11 größtes Kind von 8

|11|9|8|1|6|0|



## Aufgabe 4

(a)

- (1) 9 als Wurzelknoten einfügen

- (2) 1 unter 9 als linkes Kind einfügen, da  $1 < 9$

- (3) 8 unter 9 unter 1 (links) als rechtes Kind einfügen, da  $8 < 9$  und  $8 > 1$

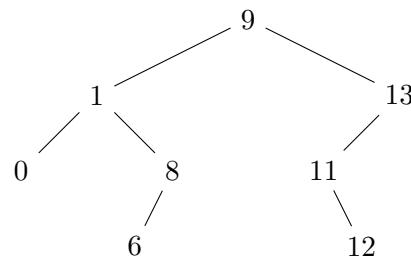
- (4) 6 unter 9 unter 1 (links) unter 8 (rechts) als linkes Kind einfügen, da  $6 < 9$ ,  $6 > 1$  und  $6 < 8$

- (5) 13 unter 9 als rechtes Kind einfügen, da  $13 > 9$

- (6) 0 unter 9 unter 1 (links) als linkes Kind einfügen, da  $0 < 9$  und  $0 < 1$

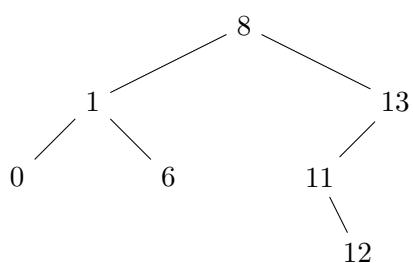
- (7) 11 unter 9 unter 13 (rechts) als linkes Kind einfügen, da  $11 > 9$  und  $11 < 13$

- (8) 12 unter 9 unter 13 (rechts) unter 11 (links) als rechts Kind einfügen, da  $12 > 9$ ,  $12 < 13$  und  $12 > 11$



(b)

- (1) 9 und 8 vertauschen
- (2) 9 entfernen, 6 fällt an die Stelle von 9



- (1) 13 entfernen, 11 (mitsamt Kind) rückt an dessen Stelle

