

29.9.2019 16:19:12

AVLTree.java

Page 1/3

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Sun Sep 29 16:19:12 CEST 2019
4  */
5
6  package uebung03.ml.aufgabe03;
7
8  import java.util.Collection;
9
10 import uebung02.ml.aufgabe01.BinarySearchTree.Entry;
11
12
13 public class AVLTree <K extends Comparable<? super K>, V> {
14
15     //private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImpl<K, V>();
16
17     // Show in ADV:
18     //private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImplADV<K, V>("AVL-Tree");
19     private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImplADV<K, V>("AVL-Tree", 1, 3);
20     // Be aware of NodeFixationException!
21
22     public V put(K key, V value) {
23         return avlTreeImpl.put(key, value);
24     }
25
26     public V get(K key) {
27         return avlTreeImpl.get(key);
28     }
29
30     public int getHeight() {
31         return avlTreeImpl.getHeight();
32     }
33
34     public int size() {
35         return avlTreeImpl.size();
36     }
37
38     public boolean isEmpty() {
39         return avlTreeImpl.isEmpty();
40     }
41
42     public void clear() {
43         avlTreeImpl.clear();
44     }
45
46     public Collection<Entry<K, V>> inorder() {
47         return avlTreeImpl.inorder();
48     }
49
50     public void printInorder() {
51         avlTreeImpl.printInorder();
52     }
53
54     public void print() {
55         avlTreeImpl.print();
56     }
57
58     protected AVLTreeImpl<K, V> getImpl() {
59         return avlTreeImpl;
60     }

```

29.9.2019 16:19:12

AVLTree.java

Page 2/3

```

61
62     public static void main(String[] args) {
63
64         AVLTree<Integer, String> avlTree = new AVLTree<Integer, String>();
65
66         System.out.println("Inserting 2:");
67         avlTree.put(2, "Str2");
68         avlTree.print();
69         System.out.println("=====");
70         System.out.println("Inserting 1:");
71         avlTree.put(1, "Str1");
72         avlTree.print();
73         System.out.println("=====");
74         System.out.println("Inserting 5:");
75         avlTree.put(5, "Str5");
76         avlTree.print();
77         System.out.println("=====");
78         System.out.println("Inserting 3:");
79         avlTree.put(3, "Str3");
80         avlTree.print();
81         System.out.println("=====");
82         System.out.println("Inserting 6:");
83         avlTree.put(6, "Str6");
84         avlTree.print();
85         System.out.println("=====");
86         System.out.println("Inserting 4:1:");
87         avlTree.put(4, "Str4:1");
88         avlTree.print();
89         System.out.println("=====");
90         System.out.println("Inserting 4:2:");
91         avlTree.put(4, "Str4:2");
92         avlTree.print();
93         System.out.println("=====");
94         System.out.println("Getting 3 : " + avlTree.get(3));
95         System.out.println("Getting 4 : " + avlTree.get(4));
96         System.out.println("Getting 7 : " + avlTree.get(7));
97
98     }
99
100 }
101

```

29.9.2019 16:19:12

AVLTree.java

Page 3/3

```

102
103 /* Session-Log:
104
105 Inserting 2:
106   2 - Str2    : h=0 ROOT
107 =====
108 Inserting 1:
109   1 - Str1    : h=0 / parent(key)=2
110   2 - Str2    : h=1 ROOT
111 =====
112 Inserting 5:
113   1 - Str1    : h=0 / parent(key)=2
114   2 - Str2    : h=1 ROOT
115   5 - Str5    : h=0 \ parent(key)=2
116 =====
117 Inserting 3:
118   1 - Str1    : h=0 / parent(key)=2
119   2 - Str2    : h=2 ROOT
120   3 - Str3    : h=0 / parent(key)=5
121   5 - Str5    : h=1 \ parent(key)=2
122 =====
123 Inserting 6:
124   1 - Str1    : h=0 / parent(key)=2
125   2 - Str2    : h=2 ROOT
126   3 - Str3    : h=0 / parent(key)=5
127   5 - Str5    : h=1 \ parent(key)=2
128   6 - Str6    : h=0 \ parent(key)=5
129 =====
130 Inserting 4:1:
131   1 - Str1    : h=0 / parent(key)=2
132   2 - Str2    : h=3 ROOT
133   3 - Str3    : h=1 / parent(key)=5
134   4 - Str4:1  : h=0 \ parent(key)=3
135   5 - Str5    : h=2 \ parent(key)=2
136   6 - Str6    : h=0 \ parent(key)=5
137 =====
138 Inserting 4:2:
139   1 - Str1    : h=0 / parent(key)=2
140   2 - Str2    : h=3 ROOT
141   3 - Str3    : h=1 / parent(key)=5
142   4 - Str4:2  : h=0 \ parent(key)=3
143   5 - Str5    : h=2 \ parent(key)=2
144   6 - Str6    : h=0 \ parent(key)=5
145 =====
146 Getting 3 :Str3
147 Getting 4 :Str4:2
148 Getting 7 :null
149
150 */

```

29.9.2019 16:19:12

AVLTreeImpl.java

Page 1/4

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Sun Sep 29 16:19:12 CEST 2019
4  */
5
6  package uebung03.ml.aufgabe03;
7
8  import java.util.Collection;
9  import java.util.LinkedList;
10 import java.util.List;
11
12 import uebung02.ml.aufgabe01.BinarySearchTree;
13
14 class AVLTreeImpl<K extends Comparable<? super K>, V> extends
15     BinarySearchTree<K, V> {
16
17     /**
18      * After the BST-operation 'insert()':
19      * actionNode shall point to the parent of the new inserted node.
20      */
21     protected AVLNode actionNode;
22
23
24     protected class AVLNode extends BinarySearchTree<K, V>.Node {
25
26         private int height;
27         private Node parent;
28
29         AVLNode(Entry<K, V> entry) {
30             super(entry);
31         }
32
33         protected AVLNode setParent(AVLNode parent) {
34             AVLNode old = avlNode(this.parent);
35             this.parent = parent;
36             return old;
37         }
38
39         protected AVLNode getParent() {
40             return avlNode(parent);
41         }
42
43         protected int setHeight(int height) {
44             int old = this.height;
45             this.height = height;
46             return old;
47         }
48
49         protected int getHeight() {
50             return height;
51         }
52
53         @Override
54         public AVLNode getLeftChild() {
55             return avlNode(super.getLeftChild());
56         }
57
58         @Override
59         public AVLNode getRightChild() {
60             return avlNode(super.getRightChild());
61         }

```

29.9.2019 16:19:12

AVLTreImpl.java

Page 2/4

```

62
63     @Override
64     public String toString() {
65         String result = String.format("%2d - %-6s : h=%d",
66                                     getEntry().getKey(), getEntry().getValue(), height);
67         if (parent == null) {
68             result += " ROOT";
69         } else {
70             boolean left = (parent.getLeftChild() == this) ? true : false;
71             result += (left ? " / " : " \\ ") + "parent(key)="
72                     + parent.getEntry().getKey();
73         }
74         return result;
75     }
76 } // End of class AVLNode
77
78
79 protected AVLNode getRoot() {
80     return avlNode(root);
81 }
82
83
84 public V put(K key, V value) {
85     Entry<K, V> entry = find(key);
86     if (entry != null) {
87         // key already exists in the Tree
88         return entry.setValue(value);
89     } else {
90         // key does not exist in the Tree yet
91         super.insert(key, value);
92         assureHeights(actionNode);
93         actionNode = null;
94         return null;
95     }
96 }
97
98 public V get(K key) {
99     Entry<K, V> entry = super.find(key);
100     if (entry != null) {
101         return entry.getValue();
102     } else {
103         return null;
104     }
105 }
106
107 @Override
108 protected Node insert(Node node, Entry<K, V> entry) {
109     if (node != null) {
110         actionNode = avlNode(node);
111     }
112     // calling now the BST-insert() which will do the work:
113     AVLNode result = avlNode(super.insert(node, entry));
114     if (node == null) {
115         // In this case: result of super.insert() is the new node!
116         result.setParent(actionNode);
117     }
118     return result;
119 }

```

29.9.2019 16:19:12

AVLTreImpl.java

Page 3/4

```

120
121 /**
122  * The height of the tree.
123  *
124  * @return The actual height. -1 for an empty tree.
125  */
126 @Override
127 public int getHeight() {
128     return height(avlNode(root));
129 }
130
131 /**
132  * Returns the height of this node.
133  *
134  * @param node
135  * @return The height or -1 if null.
136  */
137 protected int height(AVLNode node) {
138     return (node != null) ? node.getHeight() : -1;
139 }
140
141 /**
142  * Assures the heights of the tree from 'node' up to the root.
143  *
144  * @param node
145  *         The node from where to start.
146  */
147 protected void assureHeights(AVLNode node) {
148     while (node != null) {
149         setHeight(node);
150         node = node.getParent();
151     }
152 }
153
154 /**
155  * Assures the correct height for node.
156  *
157  * @param node
158  *         The node to assure its height.
159  */
160 protected void setHeight(AVLNode node) {
161     if (node == null) {
162         return;
163     }
164     int heightLeftChild = height(node.getLeftChild());
165     int heightRightChild = height(node.getRightChild());
166     node.setHeight(1 + Math.max(heightLeftChild, heightRightChild));
167 }
168
169 /**
170  * Factory-Method. Creates a new node.
171  *
172  * @param entry
173  *         The entry to be inserted in the new node.
174  * @return The new created node.
175  */
176 @Override
177 protected Node newNode(Entry<K, V> entry) {
178     return new AVLNode(entry);
179 }

```

29.9.2019 16:19:12

AVLTreeImpl.java

Page 4/4

```

180
181 /**
182  * Generates an inorder-node-list.
183  *
184  * @param nodeList
185  *       The node-list to fill in inorder.
186  * @param node
187  *       The node to start from.
188  */
189 protected void inorder(Collection<AVLNode> nodeList, AVLNode node) {
190     if (node == null)
191         return;
192     inorder(nodeList, node.getLeftChild());
193     nodeList.add(node);
194     inorder(nodeList, node.getRightChild());
195 }
196
197 // Type-Casting: Node -> AVLNode (Cast-Encapsulation)
198 @SuppressWarnings("unchecked")
199 protected AVLNode avlNode(Node node) {
200     return (AVLNode)node;
201 }
202
203 public void print() {
204     List<AVLNode> nodeList = new LinkedList<>();
205     inorder(nodeList, avlNode(root));
206     for (AVLNode node: nodeList) {
207         System.out.println(node + " ");
208     }
209 }
210
211 }
212
213

```

29.9.2019 16:19:12

AVLTreeImplADV.java

Page 1/2

```

1  /*
2   * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3   * Version: Sun Sep 29 16:19:12 CEST 2019
4   */
5
6  package uebung03.ml.aufgabe03;
7
8  import ch.hsr.adv.commons.core.logic.domain.styles.ADVStyle;
9  import ch.hsr.adv.commons.core.logic.util.ADVException;
10 import ch.hsr.adv.commons.tree.logic.domain.ADVBinaryTreeNode;
11 import ch.hsr.adv.lib.bootstrapper.ADV;
12 import ch.hsr.adv.lib.tree.logic.binarytree.BinaryTreeModule;
13
14 @SuppressWarnings("unchecked")
15 public class AVLTreeImplADV<K extends Comparable<? super K>, V>
16     extends AVLTreeImpl<K, V> {
17
18     protected BinaryTreeModule advTree;
19
20     protected class AVLNodeADV extends AVLTreeImpl<K, V>.AVLNode
21         implements ADVBinaryTreeNode<String> {
22
23         protected AVLNodeADV(Entry<K, V> entry) {
24             super(entry);
25         }
26
27         @Override
28         public String getContent() {
29             return getEntry().getKey() + " / " + getEntry().getValue();
30         }
31
32         @Override
33         public ADVStyle getStyle() {
34             return null;
35         }
36
37         @Override
38         public AVLNodeADV getLeftChild() {
39             return (AVLNodeADV) super.getLeftChild();
40         }
41
42         @Override
43         public AVLNodeADV getRightChild() {
44             return (AVLNodeADV) super.getRightChild();
45         }
46     }
47
48     // class AVLTreeImplADV.AVLNodeADV
49
50     public AVLTreeImplADV(String sessionName) {
51         this(sessionName, -1, -1);
52     }
53
54     public AVLTreeImplADV(String sessionName,
55                             int maxLeftHeight, int maxRightHeight) {
56         advTree = new BinaryTreeModule(sessionName);
57         if ((maxLeftHeight != -1) && (maxRightHeight != -1)) {
58             advTree.setFixedTreeHeight(maxLeftHeight, maxRightHeight);
59         }
60         try {
61             ADV.launch(null);
62         } catch (ADVException e) {
63             e.printStackTrace();
64             System.exit(1);
65         }
66     }

```

29.9.2019 16:19:12

AVLTreeImplADV.java

Page 2/2

```

66
67  @Override
68  protected Node newNode(Entry<K, V> entry) {
69      return new AVLNodeADV(entry);
70  }
71
72  @Override
73  public V put(K key, V value) {
74      V result = super.put(key, value);
75      displayOnADV("put(" + key + ", " + value + ")");
76      return result;
77  }
78
79  protected void displayOnADV(String advMessage) {
80      advTree.setRoot((AVLNodeADV) root);
81      try {
82          ADV.snapshot(advTree, "\n" + advMessage);
83      } catch (ADVException e) {
84          e.printStackTrace();
85          System.exit(2);
86      }
87  }
88
89  }

```

29.9.2019 16:19:12

AVLTreeJUnitTest.java

Page 1/2

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Sun Sep 29 16:19:12 CEST 2019
4  */
5
6  package uebung03.ml.aufgabe03;
7
8  import static org.junit.Assert.assertEquals;
9  import static org.junit.Assert.assertNull;
10
11  import java.util.Collection;
12  import java.util.LinkedList;
13
14  import org.junit.Before;
15  import org.junit.FixMethodOrder;
16  import org.junit.Test;
17  import org.junit.runners.MethodSorters;
18
19
20  @FixMethodOrder(MethodSorters.NAME_ASCENDING)
21  public class AVLTreeJUnitTest {
22
23      AVLTreeImpl<Integer, String> avlTree;
24
25      @Before
26      public void setUp() {
27          avlTree = new AVLTreeImpl<Integer, String>();
28      }
29
30      @Test
31      public void test01Put() {
32          int[] keys = { 2, 1, 3 };
33          String[] expected = {
34              " 1 - Str1 : h=0 / parent(key)=2",
35              " 2 - Str2 : h=1 ROOT",
36              " 3 - Str3 : h=0 \\ parent(key)=2",
37          };
38          runTest(keys, expected);
39          assertEquals(1, avlTree.getHeight());
40      }
41
42      @Test
43      public void test02Get() {
44          int[] keys = { 2, 1, 4, 5, 3 };
45          String[] expected = {
46              " 1 - Str1 : h=0 / parent(key)=2",
47              " 2 - Str2 : h=2 ROOT",
48              " 3 - Str3 : h=0 / parent(key)=4",
49              " 4 - Str4 : h=1 \\ parent(key)=2",
50              " 5 - Str5 : h=0 \\ parent(key)=4",
51          };
52          runTest(keys, expected);
53          assertEquals(2, avlTree.getHeight());
54          assertEquals("Str2", avlTree.get(2));
55          assertEquals("Str5", avlTree.get(5));
56          assertNull(avlTree.get(0));
57          assertNull(avlTree.get(6));
58      }

```

29.9.2019 16:19:12

AVLTreeJUnitTest.java

Page 2/2

```

59
60 @Test
61 public void test03() {
62     int[] keys = { 2, 3, 1 };
63     String[] expected = {
64         " 1 - Str1   : h=0 / parent(key)=2",
65         " 2 - Str2   : h=1 ROOT",
66         " 3 - Str3   : h=0 \\ parent(key)=2",
67     };
68     runTest(keys, expected);
69     assertEquals(1, avlTree.getHeight());
70     avlTree.put(2, "Str2:2");
71     avlTree.put(2, "Str2:3");
72     assertEquals(1, avlTree.getHeight());
73     expected = new String[] {
74         " 1 - Str1   : h=0 / parent(key)=2",
75         " 2 - Str2:3 : h=1 ROOT",
76         " 3 - Str3   : h=0 \\ parent(key)=2",
77     };
78     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
79     avlTree.inorder(nodes, avlTree.getRoot());
80     verify(nodes, expected);
81 }
82
83
84 private void runTest(int[] keys, String[] expected) {
85     for (int key : keys) {
86         avlTree.put(key, "Str" + key);
87     }
88     Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
89     avlTree.inorder(nodes, avlTree.getRoot());
90     assertEquals(expected.length, nodes.size());
91     verify(nodes, expected);
92 }
93
94 private void verify(Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes, String[]
expected) {
95     int i = 0;
96     for (AVLTreeImpl<Integer, String>.AVLNode node: nodes) {
97         String nodeStr = node.toString();
98         String expectedStr = expected[i];
99         assertEquals(expectedStr, nodeStr);
100         i++;
101     }
102 }
103
104 }
105

```