

30.9.2024 16:36:44

## AVLTree.java

Page 1/3

```

1  /*
2  * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3  * Version: Mon Sep 30 16:36:44 CEST 2024
4  */
5
6  package ex03.baseline.task03;
7
8  import java.util.Collection;
9
10 import ex02.solution.task01.BinarySearchTree.Entry;
11
12
13 public class AVLTree <K extends Comparable<? super K>, V> {
14
15     private AVLTreeImpl<K, V> avlTreeImpl =
16         new AVLTreeImpl<>();
17         //new AVLTreeImplADV<>("AVL-Tree"); // Show in ADV
18         //new AVLTreeImplADV<>("AVL-Tree", 1, 3); // Show in ADV: Be aware of NodeFixation
19         Exception!
20
21     public V put(K key, V value) {
22         return avlTreeImpl.put(key, value);
23     }
24
25     public V get(K key) {
26         return avlTreeImpl.get(key);
27     }
28
29     public int getHeight() {
30         return avlTreeImpl.getHeight();
31     }
32
33     public int size() {
34         return avlTreeImpl.size();
35     }
36
37     public boolean isEmpty() {
38         return avlTreeImpl.isEmpty();
39     }
40
41     public void clear() {
42         avlTreeImpl.clear();
43     }
44
45     public Collection<Entry<K, V>> inorder() {
46         return avlTreeImpl.inorder();
47     }
48
49     public void printInorder() {
50         avlTreeImpl.printInorder();
51     }
52
53     public void print() {
54         avlTreeImpl.print();
55     }
56
57     protected AVLTreeImpl<K, V> getImpl() {
58         return avlTreeImpl;
59     }

```

30.9.2024 16:36:44

## AVLTree.java

Page 2/3

```

59
60     public static void main(String[] args) {
61
62         AVLTree<Integer, String> avlTree = new AVLTree<>();
63
64         System.out.println("Inserting 2:");
65         avlTree.put(2, "Str2");
66         avlTree.print();
67         System.out.println("=====");
68         System.out.println("Inserting 1:");
69         avlTree.put(1, "Str1");
70         avlTree.print();
71         System.out.println("=====");
72         System.out.println("Inserting 5:");
73         avlTree.put(5, "Str5");
74         avlTree.print();
75         System.out.println("=====");
76         System.out.println("Inserting 3:");
77         avlTree.put(3, "Str3");
78         avlTree.print();
79         System.out.println("=====");
80         System.out.println("Inserting 6:");
81         avlTree.put(6, "Str6");
82         avlTree.print();
83         System.out.println("=====");
84         System.out.println("Inserting 4:1:");
85         avlTree.put(4, "Str4:1");
86         avlTree.print();
87         System.out.println("=====");
88         System.out.println("Inserting 4:2:");
89         avlTree.put(4, "Str4:2");
90         avlTree.print();
91         System.out.println("=====");
92         System.out.println("Getting 3 : " + avlTree.get(3));
93         System.out.println("Getting 4 : " + avlTree.get(4));
94         System.out.println("Getting 7 : " + avlTree.get(7));
95
96     }
97
98 }
99

```

30.9.2024 16:36:44

## AVLTree.java

Page 3/3

```

100
101 /* Session-Log:
102
103 Inserting 2:
104 2 - Str2 : h=0 ROOT
105 =====
106 Inserting 1:
107 1 - Str1 : h=0 / parent(key)=2
108 2 - Str2 : h=1 ROOT
109 =====
110 Inserting 5:
111 1 - Str1 : h=0 / parent(key)=2
112 2 - Str2 : h=1 ROOT
113 5 - Str5 : h=0 \ parent(key)=2
114 =====
115 Inserting 3:
116 1 - Str1 : h=0 / parent(key)=2
117 2 - Str2 : h=2 ROOT
118 3 - Str3 : h=0 / parent(key)=5
119 5 - Str5 : h=1 \ parent(key)=2
120 =====
121 Inserting 6:
122 1 - Str1 : h=0 / parent(key)=2
123 2 - Str2 : h=2 ROOT
124 3 - Str3 : h=0 / parent(key)=5
125 5 - Str5 : h=1 \ parent(key)=2
126 6 - Str6 : h=0 \ parent(key)=5
127 =====
128 Inserting 4:1:
129 1 - Str1 : h=0 / parent(key)=2
130 2 - Str2 : h=3 ROOT
131 3 - Str3 : h=1 / parent(key)=5
132 4 - Str4:1 : h=0 \ parent(key)=3
133 5 - Str5 : h=2 \ parent(key)=2
134 6 - Str6 : h=0 \ parent(key)=5
135 =====
136 Inserting 4:2:
137 1 - Str1 : h=0 / parent(key)=2
138 2 - Str2 : h=3 ROOT
139 3 - Str3 : h=1 / parent(key)=5
140 4 - Str4:2 : h=0 \ parent(key)=3
141 5 - Str5 : h=2 \ parent(key)=2
142 6 - Str6 : h=0 \ parent(key)=5
143 =====
144 Getting 3 :Str3
145 Getting 4 :Str4:2
146 Getting 7 :null
147
148 */

```

30.9.2024 16:36:44

## AVLTreeImpl.java

Page 1/3

```

1  /*
2  * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3  * Version: Mon Sep 30 16:36:44 CEST 2024
4  */
5
6  package ex03.baseline.task03;
7
8  import java.util.Collection;
9  import java.util.LinkedList;
10 import java.util.List;
11
12 import ex02.solution.task01.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         }

```

30.9.2024 16:36:44

AVLTreImpl.java

Page 2/3

```

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     // TODO Implement here...
86     return null;
87 }
88
89 public V get(K key) {
90     // TODO Implement here...
91     return null;
92 }
93
94 @Override
95 protected Node insert(Node node, Entry<K, V> entry) {
96     // TODO Implement here...
97     return null;
98 }
99
100 /**
101  * The height of the tree.
102  *
103  * @return The current height. -1 for an empty tree.
104  */
105 @Override
106 public int getHeight() {
107     return height(avlNode(root));
108 }
109
110 /**
111  * Returns the height of this node.
112  *
113  * @param node
114  * @return The height or -1 if null.
115  */
116 @SuppressWarnings("static-method")
117 protected int height(AVLNode node) {
118     return (node != null) ? node.getHeight() : -1;
119 }
120
121 /**
122  * Assures the heights of the tree from 'node' up to the root.
123  *
124  * @param node
125  *     The node from where to start.
126  */
127 protected void assureHeights(AVLNode node) {
128     // TODO Implement here...
129 }

```

30.9.2024 16:36:44

AVLTreImpl.java

Page 3/3

```

130
131 /**
132  * Assures the correct height for node.
133  *
134  * @param node
135  *     The node to assure its height.
136  */
137 protected void setHeight(AVLNode node) {
138     // TODO Implement here...
139 }
140
141 /**
142  * Factory-Method. Creates a new node.
143  *
144  * @param entry
145  *     The entry to be inserted in the new node.
146  * @return The new created node.
147  */
148 @Override
149 protected Node newNode(Entry<K, V> entry) {
150     // TODO Implement here...
151     return null;
152 }
153
154 @Override
155 protected void inorder(Node node, Collection<Node> inorderList) {
156     super.inorder(node, inorderList);
157 }
158
159 // Type-Casting: Node -> AVLNode (Cast-Encapsulation)
160 @SuppressWarnings({ "unchecked", "static-method" })
161 protected AVLNode avlNode(Node node) {
162     return (AVLNode)node;
163 }
164
165 public void print() {
166     List<Node> nodeList = new LinkedList<>();
167     inorder(root, nodeList);
168     for (Node node: nodeList) {
169         System.out.println(node + " ");
170     }
171 }
172
173 }
174
175

```

30.9.2024 16:36:44

AVLTreeImplADV.java

Page 1/2

```

1  /*
2   * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3   * Version: Mon Sep 30 16:36:44 CEST 2024
4   */
5
6  package ex03.baseline.task03;
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() + " (" + getHeight()
30 + ")";
31         }
32
33         @Override
34         public ADVStyle getStyle() {
35             return null;
36         }
37
38         @Override
39         public AVLNodeADV getLeftChild() {
40             return (AVLNodeADV) super.getLeftChild();
41         }
42
43         @Override
44         public AVLNodeADV getRightChild() {
45             return (AVLNodeADV) super.getRightChild();
46         }
47     } // class AVLTreeImplADV.AVLNodeADV
48
49     public AVLTreeImplADV(String sessionName) {
50         this(sessionName, -1, -1);
51     }
52
53     public AVLTreeImplADV(String sessionName,
54         int maxLeftHeight, int maxRightHeight) {
55         advTree = new BinaryTreeModule(sessionName);
56         if ((maxLeftHeight != -1) && (maxRightHeight != -1)) {
57             advTree.setFixedTreeHeight(maxLeftHeight, maxRightHeight);
58         }
59         try {
60             ADV.launch(null);
61         } catch (ADVException e) {
62             e.printStackTrace();
63             System.exit(1);
64         }
65     }

```

30.9.2024 16:36:44

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 }

```

30.9.2024 16:36:44

## AVLTreeJUnitTest.java

Page 1/2

```

1  /*
2   * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3   * Version: Mon Sep 30 16:36:44 CEST 2024
4   */
5
6  package ex03.baseline.task03;
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  import ex02.solution.task01.BinarySearchTree;
20
21  @FixMethodOrder(MethodSorters.NAME_ASCENDING)
22  public class AVLTreeJUnitTest {
23
24
25      AVLTreeImpl<Integer, String> avlTree;
26
27      @Before
28      public void setUp() {
29          avlTree = new AVLTreeImpl<>();
30      }
31
32      @Test
33      public void test01Put() {
34          int[] keys = { 2, 1, 3 };
35          String[] expected = {
36              " 1 - Str1   : h=0 / parent(key)=2",
37              " 2 - Str2   : h=1 ROOT",
38              " 3 - Str3   : h=0 \\ parent(key)=2",
39          };
40          runTest(keys, expected);
41          assertEquals(1, avlTree.getHeight());
42      }
43
44      @Test
45      public void test02Get() {
46          int[] keys = { 2, 1, 4, 5, 3 };
47          String[] expected = {
48              " 1 - Str1   : h=0 / parent(key)=2",
49              " 2 - Str2   : h=2 ROOT",
50              " 3 - Str3   : h=0 / parent(key)=4",
51              " 4 - Str4   : h=1 \\ parent(key)=2",
52              " 5 - Str5   : h=0 \\ parent(key)=4",
53          };
54          runTest(keys, expected);
55          assertEquals(2, avlTree.getHeight());
56          assertEquals("Str2", avlTree.get(2));
57          assertEquals("Str5", avlTree.get(5));
58          assertNull(avlTree.get(0));
59          assertNull(avlTree.get(6));
60      }

```

30.9.2024 16:36:44

## AVLTreeJUnitTest.java

Page 2/2

```

61
62  @Test
63  public void test03() {
64      int[] keys = { 2, 3, 1 };
65      String[] expected = {
66          " 1 - Str1   : h=0 / parent(key)=2",
67          " 2 - Str2   : h=1 ROOT",
68          " 3 - Str3   : h=0 \\ parent(key)=2",
69      };
70      runTest(keys, expected);
71      assertEquals(1, avlTree.getHeight());
72      avlTree.put(2, "Str2:2");
73      avlTree.put(2, "Str2:3");
74      assertEquals(1, avlTree.getHeight());
75      expected = new String[] {
76          " 1 - Str1   : h=0 / parent(key)=2",
77          " 2 - Str2:3 : h=1 ROOT",
78          " 3 - Str3   : h=0 \\ parent(key)=2",
79      };
80      Collection<BinarySearchTree<Integer, String>.Node> nodes = new LinkedList<>();
81      avlTree.inorder(avlTree.getRoot(), nodes);
82      verify(nodes, expected);
83  }
84
85
86  private void runTest(int[] keys, String[] expected) {
87      for (int key : keys) {
88          avlTree.put(key, "Str" + key);
89      }
90      Collection<BinarySearchTree<Integer, String>.Node> nodes = new LinkedList<>();
91      avlTree.inorder(avlTree.getRoot(), nodes);
92      assertEquals(expected.length, nodes.size());
93      verify(nodes, expected);
94  }
95
96  private static void verify(Collection<BinarySearchTree<Integer, String>.Node> nodes,
97      String[] expected) {
98      int i = 0;
99      for (BinarySearchTree<Integer, String>.Node node: nodes) {
100          String nodeStr = node.toString();
101          String expectedStr = expected[i];
102          assertEquals(expectedStr, nodeStr);
103          i++;
104      }
105  }
106  }
107

```