

1.10.2018 20:09:33

AVLTree.java

Page 1/3

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Mon Oct 1 20:09:33 CEST 2018
4  */
5
6  package uebung03.as.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     // Start the GVS-Server first: Double-Click 'GVS_Server_v1.4.jar'
18     //private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImplGVS<K, V>();
19
20     public V put(K key, V value) {
21         return avlTreeImpl.put(key, value);
22     }
23
24     public V get(K key) {
25         return avlTreeImpl.get(key);
26     }
27
28     public int getHeight() {
29         return avlTreeImpl.getHeight();
30     }
31
32     public int size() {
33         return avlTreeImpl.size();
34     }
35
36     public boolean isEmpty() {
37         return avlTreeImpl.isEmpty();
38     }
39
40     public void clear() {
41         avlTreeImpl.clear();
42     }
43
44     public Collection<Entry<K, V>> inorder() {
45         return avlTreeImpl.inorder();
46     }
47
48     public void printInorder() {
49         avlTreeImpl.printInorder();
50     }
51
52     public void print() {
53         avlTreeImpl.print();
54     }
55
56     protected AVLTreeImpl<K, V> getImpl() {
57         return avlTreeImpl;
58     }

```

1.10.2018 20:09:33

AVLTree.java

Page 2/3

```

59
60     public static void main(String[] args) {
61
62         AVLTree<Integer, String> avlTree = new AVLTree<Integer, String>();
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         if (avlTree.getImpl() instanceof AVLTreeImplGVS) {
97             ((AVLTreeImplGVS<Integer, String>)avlTree.getImpl()).gvsTree.disconnect();
98         }
99
100     }
101
102 }
103

```

1.10.2018 20:09:33

## AVLTree.java

Page 3/3

```

104
105 /* Session-Log:
106
107 Inserting 2:
108   2 - Str2   : h=0 ROOT
109 =====
110 Inserting 1:
111   1 - Str1   : h=0 / parent(key)=2
112   2 - Str2   : h=1 ROOT
113 =====
114 Inserting 5:
115   1 - Str1   : h=0 / parent(key)=2
116   2 - Str2   : h=1 ROOT
117   5 - Str5   : h=0 \ parent(key)=2
118 =====
119 Inserting 3:
120   1 - Str1   : h=0 / parent(key)=2
121   2 - Str2   : h=2 ROOT
122   3 - Str3   : h=0 / parent(key)=5
123   5 - Str5   : h=1 \ parent(key)=2
124 =====
125 Inserting 6:
126   1 - Str1   : h=0 / parent(key)=2
127   2 - Str2   : h=2 ROOT
128   3 - Str3   : h=0 / parent(key)=5
129   5 - Str5   : h=1 \ parent(key)=2
130   6 - Str6   : h=0 \ parent(key)=5
131 =====
132 Inserting 4:1:
133   1 - Str1   : h=0 / parent(key)=2
134   2 - Str2   : h=3 ROOT
135   3 - Str3   : h=1 / parent(key)=5
136   4 - Str4:1 : h=0 \ parent(key)=3
137   5 - Str5   : h=2 \ parent(key)=2
138   6 - Str6   : h=0 \ parent(key)=5
139 =====
140 Inserting 4:2:
141   1 - Str1   : h=0 / parent(key)=2
142   2 - Str2   : h=3 ROOT
143   3 - Str3   : h=1 / parent(key)=5
144   4 - Str4:2 : h=0 \ parent(key)=3
145   5 - Str5   : h=2 \ parent(key)=2
146   6 - Str6   : h=0 \ parent(key)=5
147 =====
148 Getting 3 :Str3
149 Getting 4 :Str4:2
150 Getting 7 :null
151
152 */

```

1.10.2018 20:09:33

## AVLTreeImpl.java

Page 1/3

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Mon Oct 1 20:09:33 CEST 2018
4  */
5
6  package uebung03.as.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         }

```

1.10.2018 20:09:33

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 actual 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 protected int height(AVLNode node) {
117     return (node != null) ? node.getHeight() : -1;
118 }
119
120 /**
121  * Assures the heights of the tree from 'node' up to the root.
122  *
123  * @param node
124  *         The node from where to start.
125  */
126 protected void assureHeights(AVLNode node) {
127     // TODO Implement here...
128 }

```

1.10.2018 20:09:33

AVLTreImpl.java

Page 3/3

```

129
130 /**
131  * Assures the correct height for node.
132  *
133  * @param node
134  *         The node to assure its height.
135  */
136 protected void setHeight(AVLNode node) {
137     // TODO Implement here...
138 }
139
140 /**
141  * Factory-Method. Creates a new node.
142  *
143  * @param entry
144  *         The entry to be inserted in the new node.
145  * @return The new created node.
146  */
147 @Override
148 protected Node newNode(Entry<K, V> entry) {
149     // TODO Implement here...
150     return null;
151 }
152
153 /**
154  * Generates an inorder-node-list.
155  *
156  * @param nodeList
157  *         The node-list to fill in inorder.
158  * @param node
159  *         The node to start from.
160  */
161 protected void inorder(Collection<AVLNode> nodeList, AVLNode node) {
162     if (node == null)
163         return;
164     inorder(nodeList, node.getLeftChild());
165     nodeList.add(node);
166     inorder(nodeList, node.getRightChild());
167 }
168
169 // Type-Casting: Node -> AVLNode (Cast-Encapsulation)
170 @SuppressWarnings("unchecked")
171 protected AVLNode avlNode(Node node) {
172     return (AVLNode)node;
173 }
174
175 public void print() {
176     List<AVLNode> nodeList = new LinkedList<>();
177     inorder(nodeList, avlNode(root));
178     for (AVLNode node: nodeList) {
179         System.out.println(node + " ");
180     }
181 }
182
183 }
184
185

```

1.10.2018 20:09:33

AVLTreeImplGVS.java

Page 1/2

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Mon Oct 1 20:09:33 CEST 2018
4  */
5
6  package uebung03.as.aufgabe03;
7
8  import gvs.tree.GVSBinaryTreeNode;
9  import gvs.tree.GVSTreeWithRoot;
10 import gvs.typ.node.GVSNodeType;
11
12 class AVLTreeImplGVS<K extends Comparable<? super K>, V> extends
13     AVLTreeImpl<K, V> {
14
15     protected GVSTreeWithRoot gvsTree;
16
17     private final int DELAY = 200;
18
19     protected class AVLNodeGVS extends AVLTreeImpl<K, V>.AVLNode implements GVSBinaryTree
20     eNode {
21
22         protected AVLNodeGVS(Entry<K, V> entry) {
23             super(entry);
24         }
25
26         public GVSBinaryTreeNode getGVSBinaryLeftChild() {
27             return (GVSBinaryTreeNode) getLeftChild();
28         }
29
30         public GVSBinaryTreeNode getGVSBinaryRightChild() {
31             return (GVSBinaryTreeNode) getRightChild();
32         }
33
34         public String getNodeLabel() {
35             Entry<K, V> e = getEntry();
36             return e.getKey() + " " + e.getValue();
37             //return e.getKey().toString();
38         }
39
40         public GVSNodeType getNodeTyp() {
41             return null;
42         }
43     } // class BinaryTreeTestGVS.NodeGVS
44
45     AVLTreeImplGVS() {
46         this("AVLTreeGVS");
47     }
48
49     AVLTreeImplGVS(String title) {
50         gvsTree = new GVSTreeWithRoot(title);
51     }
52
53     @Override
54     protected Node newNode(Entry<K, V> entry) {
55         return new AVLNodeGVS(entry);
56     }
57
58     @Override
59     public V put(K key, V value) {
60         V result = super.put(key, value);
61         gvsTree.setRoot((GVSBinaryTreeNode) root);
62         gvsTree.display();
63         try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
64         return result;
65     }
66
67

```

1.10.2018 20:09:33

AVLTreeImplGVS.java

Page 2/2

```

68
69     @Override
70     public Entry<K, V> insert(K key, V value) {
71         Entry<K, V> newEntry = super.insert(key, value);
72         gvsTree.setRoot((GVSBinaryTreeNode) root);
73         gvsTree.display();
74         try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
75         return newEntry;
76     }
77
78     @Override
79     public Entry<K, V> remove(Entry<K, V> entry) {
80         Entry<K, V> deletedEntry = super.remove(entry);
81         gvsTree.display();
82         try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
83         return deletedEntry;
84     }
85
86 }
87
88
89

```

1.10.2018 20:09:33

## AVLTreeJUnitTest.java

Page 1/2

```

1  /*
2   * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3   * Version: Mon Oct 1 20:09:33 CEST 2018
4   */
5
6  package uebung03.as.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.After;
15 import org.junit.Before;
16 import org.junit.FixMethodOrder;
17 import org.junit.Test;
18 import org.junit.runners.MethodSorters;
19
20
21 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
22 public class AVLTreeJUnitTest {
23
24     AVLTreeImpl<Integer, String> avlTree;
25
26     @Before
27     public void setUp() {
28         //System.setProperty("NoGVS", "true");
29         avlTree = new AVLTree<Integer, String>().getImpl();
30     }
31
32     @After
33     public void tearDown() {
34         if (avlTree instanceof AVLTreeImplGVS) {
35             ((AVLTreeImplGVS<Integer, String>)avlTree).gvsTree.disconnect();
36         }
37     }
38
39     @Test
40     public void test01Put() {
41         int[] keys = { 2, 1, 3 };
42         String[] expected = {
43             " 1 - Str1 : h=0 / parent(key)=2",
44             " 2 - Str2 : h=1 ROOT",
45             " 3 - Str3 : h=0 \\ parent(key)=2",
46         };
47         runTest(keys, expected);
48         assertEquals(1, avlTree.getHeight());
49     }
50
51     @Test
52     public void test02Get() {
53         int[] keys = { 2, 1, 4, 5, 3 };
54         String[] expected = {
55             " 1 - Str1 : h=0 / parent(key)=2",
56             " 2 - Str2 : h=2 ROOT",
57             " 3 - Str3 : h=0 / parent(key)=4",
58             " 4 - Str4 : h=1 \\ parent(key)=2",
59             " 5 - Str5 : h=0 \\ parent(key)=4",
60         };
61         runTest(keys, expected);
62         assertEquals(2, avlTree.getHeight());
63         assertEquals("Str2", avlTree.get(2));
64         assertEquals("Str5", avlTree.get(5));
65         assertNull(avlTree.get(0));
66         assertNull(avlTree.get(6));
67     }

```

1.10.2018 20:09:33

## AVLTreeJUnitTest.java

Page 2/2

```

68
69     @Test
70     public void test03() {
71         int[] keys = { 2, 3, 1 };
72         String[] expected = {
73             " 1 - Str1 : h=0 / parent(key)=2",
74             " 2 - Str2 : h=1 ROOT",
75             " 3 - Str3 : h=0 \\ parent(key)=2",
76         };
77         runTest(keys, expected);
78         assertEquals(1, avlTree.getHeight());
79         avlTree.put(2, "Str2:2");
80         avlTree.put(2, "Str2:3");
81         assertEquals(1, avlTree.getHeight());
82         expected = new String[] {
83             " 1 - Str1 : h=0 / parent(key)=2",
84             " 2 - Str2:3 : h=1 ROOT",
85             " 3 - Str3 : h=0 \\ parent(key)=2",
86         };
87         Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
88         avlTree.inorder(nodes, avlTree.getRoot());
89         verify(nodes, expected);
90     }
91
92
93     private void runTest(int[] keys, String[] expected) {
94         for (int key : keys) {
95             avlTree.put(key, "Str" + key);
96         }
97         Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
98         avlTree.inorder(nodes, avlTree.getRoot());
99         assertEquals(expected.length, nodes.size());
100        verify(nodes, expected);
101    }
102
103    private void verify(Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes, String[]
104    expected) {
105        int i = 0;
106        for (AVLTreeImpl<Integer, String>.AVLNode node: nodes) {
107            String nodeStr = node.toString();
108            String expectedStr = expected[i];
109            assertEquals(expectedStr, nodeStr);
110            i++;
111        }
112    }
113
114

```