```
BinarySearchTree.java
24.9.2018 12:08:35
                                                                                   Page 1/4
    * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
    * Version: Mon Sep 24 12:08:35 CEST 2018
3
   package uebung02.as.aufgabe01;
   import java.util.Collection;
   import java.util.LinkedList;
   public class BinarySearchTree<K extends Comparable<? super K>, V> {
12
13
     protected Node root;
14
     public static class Entry<K, V> {
16
17
        private K key;
        private V value;
18
        public Entry(K key, V value) {
20
21
          this.key = key;
          this.value = value;
22
23
24
        protected K setKey(K key) {
25
         K oldKey = this.key;
26
          this.key = key;
27
          return oldKey;
28
29
30
        public K getKey() {
31
32
          return key;
33
34
        public V setValue(V value) {
35
          V oldValue = this.value;
          this.value = value;
37
          return oldValue;
38
39
        public V getValue() {
41
          return value;
42
43
45
        @Override
46
        public String toString() {
47
          StringBuilder result = new StringBuilder();
          result.append("[").append(key).append("/").append(value).append("]");
          return result.toString();
     } // End of class Entry
52
     protected class Node {
55
        private Entry<K, V> entry;
56
        private Node leftChild;
        private Node rightChild;
58
59
        public Node(Entry<K, V> entry) {
60
          this.entry = entry;
61
62
63
        public Node(Entry<K, V> entry, Node leftChild, Node rightChild) {
64
          this.entry = entry;
65
          this.leftChild = leftChild;
          this.rightChild = rightChild;
67
68
```

```
BinarySearchTree.java
24.9.2018 12:08:35
                                                                                    Page 2/4
70
        public Entry<K, V> getEntry() {
         return entry;
71
72
73
        public Entry<K, V> setEntry(Entry<K, V> entry) {
          Entry<K, V> oldEntry = entry;
75
76
          this.entry = entry;
77
          return oldEntry;
78
80
        public Node getLeftChild() {
81
         return leftChild;
82
83
        public void setLeftChild(Node leftChild) {
84
85
          this.leftChild = leftChild;
86
       public Node getRightChild()
88
89
         return rightChild;
90
91
        public void setRightChild(Node rightChild) {
92
93
          this.rightChild = rightChild;
94
95
     } // End of class Node
97
98
     public Entry<K, V> insert(K key, V value) {
       // TODO Implement here...
qq
100
       return null;
101
102
103
104
      * Factory-Method: Creates a new node.
105
106
       * @param entry
                  The entry to be inserted in the new node.
107
108
       * @return The new created node.
109
     protected Node newNode(Entry<K, V> entry) {
110
       return new Node(entry);
111
112
113
114
     public void clear()
115
       // TODO Implement here...
116
117
     public Entry<K, V> find(K key) {
118
119
       // TODO Implement here...
       return null;
120
121
122
123
       * Returns a collection with all entries with key.
124
125
       * @param key
126
127
                  The key to be searched.
        @return Collection of all entries found. An empty collection is returned if
128
                 no entry with key is found.
129
130
     public Collection<Entry<K, V>> findAll(K key) {
131
132
       // TODO Implement here...
       return null;
133
```

```
BinarySearchTree.java
24.9.2018 12:08:35
                                                                                      Page 3/4
135
136
       * Returns a collection with all entries in inorder.
137
138
       * @return Inorder-Collection of all entries.
139
140
      public Collection<Entry<K, V>> inorder()
141
        // TODO Implement here...
142
143
        return null;
144
146
      * Prints the entries of the tree as a list in inorder to the console.
147
148
      public void printInorder() {
150
        // TODO Implement here...
151
152
      public Entry<K, V> remove(Entry<K, V> entry) {
153
       // TODO Implement here...
154
155
        return null;
156
157
158
       * The height of the tree.
159
160
       * @return The actual height. -1 for an empty tree.
161
162
      public int getHeight() {
163
164
        // TODO Implement here...
        return -1;
165
167
168
      public int size() {
        // TODO Implement here...
169
        return -1;
171
172
      public boolean isEmpty() {
173
174
        // TODO Implement here...
175
        return true;
176
```

```
BinarySearchTree.java
24.9.2018 12:08:35
                                                                                      Page 4/4
     public static void main(String[] args) {
179
180
        // Example from lecture "L?schen (IV/IV)":
        BinarySearchTree<Integer, String> bst = new BinarySearchTree<>();
181
        //BinarySearchTree<Integer, String> bst = new BinarySearchTreeGVS<>();
        System.out.println("Inserting:");
183
184
        bst.insert(1, "Str1");
185
       bst.printInorder();
        bst.insert(3, "Str3");
186
        bst.printInorder();
187
188
        bst.insert(2, "Str2");
189
        bst.printInorder();
        bst.insert(8, "Str8");
190
191
        bst.printInorder();
192
        bst.insert(9, "Str9");
193
        bst.insert(6, "Str6");
        bst.insert(5, "Str5");
194
        bst.printInorder();
196
197
        System.out.println("Removeing 3:");
        Entry<Integer, String> entry = bst.find(3);
198
        System.out.println(entry);
199
        bst.remove(entry);
200
201
        bst.printInorder();
202
        //if (bst instanceof BinarySearchTreeGVS) {
203
        // ((BinarySearchTreeGVS<Integer, String>)bst).gvsTree.disconnect();
204
205
        //}
206
207
208
     /* Session-Log:
209
210
     Inserting:
211
     [1/Str1]
     [1/Str1] [3/Str3]
213
     [1/Str1] [2/Str2] [3/Str3] [1/Str1] [2/Str2] [3/Str3] [8/Str8]
215
     [1/Str1] [2/Str2] [3/Str3] [5/Str5] [6/Str6] [8/Str8] [9/Str9]
      [3/Str3]
218
     [1/Str1] [2/Str2] [5/Str5] [6/Str6] [8/Str8] [9/Str9]
219
220
221
222
     // End of class BinarySearchTree
224
```

BinarySearchTreeTest.java 24.9.2018 12:08:35 Page 1/2 * HSR - Uebungen 'Algorithmen & Datenstrukturen 2' * Version: Mon Sep 24 12:08:35 CEST 2018 3 package uebung02.as.aufgabe01; import java.util.Iterator; 8 import java.util.Random; import uebung02.as.aufgabe01.BinarySearchTree.Entry; 12 public class BinarySearchTreeTest { 13 private static Random randomGenerator = new Random(1); 16 17 private static BinarySearchTree<Integer, String> generateTree(int nodes) { 18 BinarySearchTree<Integer, String> ret = new BinarySearchTree<>(); for (int i = 0; i < nodes; i++) { key = randomGenerator.nextInt() * Integer.MAX VALUE;</pre> 20 21 ret.insert(key, "String_" + i); 22 23 24 return ret; 25 26 public static void main(String[] args) 27 System.out.println("BINARY TREE TEST"); 28 System.out 29 30 .println("Please be patient, the following operations may take some time..."); final int TESTRUNS = 100; final int BEGINSIZE = 10000; final int VARYSIZE = 10; 33 34 long startTime = System.currentTimeMillis(); 35 BinarySearchTree<Integer, String> bst = new BinarySearchTree<>(); double avgHeight = 0; 37 double avgEntries = 0; double avgTime = 0; 39 for (int i = 0; i < TESTRUNS; i++) startTime = System.currentTimeMillis(); bst = generateTree(BEGINSIZE + i * VARYSIZE); 42 avgTime += System.currentTimeMillis() - startTime; 43 avgHeight += bst.getHeight(); avgEntries += BEGINSIZE + i * VARYSIZE; 45 46 47 avgTime /= TESTRUNS; avgEntries /= TESTRUNS; avgHeight /= TESTRUNS; System.out.println("Test successful, results are as follows:"); 50 System.out.println("Average time for generation is: " + avgTime + "ms"); 51 System.out.println("Average entries are: " + avgEntries); 52 System.out.println("Average height is: " + avgHeight); System.out.println("In h=C*log2(n), C=h/log2(n) = " + avgHeight54 55 / (Math.log(avgEntries) / Math.log(2))); System.out.println();

```
BinarySearchTreeTest.java
24.9.2018 12:08:35
                                                                                  Page 2/2
       bst = generateTree(20);
       int search = 15138431;
50
60
       Entry<Integer, String> searchResult;
       bst.insert(search, "String_" + search);
61
        searchResult = bst.find(search);
       if (searchResult == null)
63
64
         System.err.println("Search for node " + search + " failed!");
65
         System.out.println("Search for node " + search + " successful!");
66
67
68
        System.out.println();
       bst.insert(search, "String_" + search);
69
       bst.insert(search, "String_" + search);
70
       bst.insert(search, "String" + search);
72
       Iterator<Entry<Integer, String>> it = bst.findAll(search).iterator();
       int count = 0;
        while (it.hasNext())
74
         count++;
         it.next();
76
         System.out.println("Search for node " + search + " successful!");
77
78
        System.out.println("Search for node " + search + ": " + count
79
           + " nodes found!");
80
81
        System.out.println();
       it = bst.findAll(search).iterator();
82
       count = 0;
        while (it.hasNext())
84
         bst.remove(it.next());
85
       it = bst.findAll(search).iterator();
        count = 0;
89
90
        while (it.hasNext()) {
         count++;
91
         it.next();
         System.out.println("Search for node " + search + " successful!");
93
94
        System.out.println("Search for node " + search + ": " + count
95
96
            + " nodes found!");
97
98
99
100
101
   /* Session-Log:
102
   BINARY TREE TEST
   Please be patient, the following operations may take some time...
   Test successful, results are as follows:
   Average time for generation is: 9.07ms
   Average entries are: 10495.0
108 Average height is: 30.81
   In h=C*log2(n), C=h/log2(n) = 2.306584099301782
111 Search for node 15138431 successful!
113 Search for node 15138431 successful!
114 Search for node 15138431 successful!
   Search for node 15138431 successful!
116 Search for node 15138431 successful!
117 Search for node 15138431: 4 nodes found!
119 Search for node 15138431: 0 nodes found!
120
121
```

BinarySearchTreeJUnitTest.java 24.9.2018 12:08:35 Page 1/4 * HSR - Uebungen 'Algorithmen & Datenstrukturen 2' * Version: Mon Sep 24 12:08:35 CEST 2018 3 4 package uebung02.as.aufgabe01; import static org.junit.Assert.*; import java.util.Collection; import java.util.HashMap; import java.util.LinkedList; import java.util.List; import java.util.Map; import java.util.Random; import org.junit.Before; import org.junit.FixMethodOrder; import org.junit.Test; import org.junit.runners.MethodSorters; import uebung02.as.aufgabe01.BinarySearchTree.Entry; 22 @FixMethodOrder(MethodSorters.NAME ASCENDING) 25 public class BinarySearchTreeJUnitTest { BinarySearchTree<Integer, String> bst; 27 28 @Before 29 30 public void setUp() { bst = new BinarySearchTree<Integer, String>(); 33 34 @Test public void test01EmptySizeInsertClear() { 35 assertTrue(bst.isEmpty()); 37 assertEquals(0, bst.size()); bst.insert(1, "String_1"); assertEquals(1, bst.size()); 38 39 assertFalse(bst.isEmpty()); bst.insert(2, "String_2"); assertEquals(2, bst.size()); 42 bst.insert(2, "String_2"); 43 assertEquals(3, bst.size()); 45 bst.clear(); 46 assertTrue(bst.isEmpty()); 47 assertEquals(0, bst.size()); @Test 50 51 public void test02Find() Entry<Integer, String> entry; 52 entry = bst.find(1); assertNull(entry); 54 Entry<Integer, String> insertedEntry = bst.insert(1, "String_1"); 56 entry = bst.find(1); assertNotNull(entry); assertEquals(Integer.valueOf(1), entry.getKey()); assertEquals("String_1", entry.getValue()); 59 60 assertSame(insertedEntry, entry);

```
BinarySearchTreeJUnitTest.java
24.9.2018 12:08:35
                                                                                   Page 2/4
63
     public void test03FindAll() {
64
65
       Collection<Entry<Integer, String>> col;
       col = bst.findAll(1);
66
       assertEquals(0, col.size());
       bst.insert(1, "String_1");
68
69
       col = bst.findAll(2);
70
       assertEquals(0, col.size());
       bst.insert(2, "String_2");
71
       col = bst.findAll(2);
       assertEquals(1, col.size());
73
74
       bst.insert(2, "String_2");
       col = bst.findAll(2);
75
       assertEquals(2, col.size());
77
78
79
     @Test.
     public void test04GetHeight()
       assertEquals(-1, bst.getHeight());
81
       bst.insert(1, "String 1");
82
       assertEquals(0, bst.getHeight());
83
       bst.insert(2, "String 2");
84
       assertEquals(1, bst.getHeight());
85
86
87
88
     @Test
89
     public void test05Remove() {
       Entry<Integer, String> entry = new Entry<>(1, "String 1");
90
91
       entry = bst.remove(entry);
       assertNull(entry);
92
        final Entry<Integer, String> entry1 = bst.insert(1, "String_1");
        entry = bst.remove(entry1);
94
       assertSame(entry, entry1);
95
       assertEquals(0, bst.size());
96
        final Entry<Integer, String> entryla = bst.insert(1, "String_la");
       final Entry<Integer, String> entrylb = bst.insert(1, "String_1b");
98
        assertEquals(2, bst.size());
99
       entry = bst.remove(entryla);
100
       assertSame(entryla, entry);
102
       assertEquals(1, bst.size());
103
        entry = bst.remove(entry1b);
104
       assertSame(entrylb, entry);
105
       assertEquals(0, bst.size());
106
```

BinarySearchTreeJUnitTest.java 24.9.2018 12:08:35 Page 3/4 @Test public void test06RemoveCase3() { 100 110 bst.insert(1, "String_1"); Entry<Integer, String> entryToRemove = bst.insert(3, "String_3"); 111 bst.insert(2, "String 2"); 112 bst.insert(8, "String_8"); 113 bst.insert(6, "String_6"); bst.insert(9, "String_9"); 114 115 bst.insert(5, "String 5"); 116 117 assertEquals(7, bst.size()); 118 assertEquals(4, bst.getHeight()); 119 Entry<Integer, String> removedEntry = bst.remove(entryToRemove); assertSame(entryToRemove, removedEntry); 120 121 assertEquals(6, bst.size()); assertEquals(3, bst.getHeight()); 122 123 bst.remove(bst.find(6)); assertEquals(5, bst.size()); 124 assertEquals(3, bst.getHeight()); bst.remove(bst.find(9)); 126 127 assertEquals(4, bst.size()); assertEquals(2, bst.getHeight()); 128 129 130 131 @Test public void test07RemoveCase3Special() { 132 bst.insert(2, "String 2"); 133 134 bst.insert(1, "String_1"); bst.insert(3, "String_3.1"); bst.insert(3, "String_3.2"); 135 136 Collection<Entry<Integer, String>> col; 137 col = bst.findAll(3); assertEquals(2, col.size()); 139 140 Entry<Integer, String> removedEntry = bst.remove(bst.find(2)); assertNotNull(removedEntry); 141 assertEquals("String_2", removedEntry.getValue()); col = bst.findAll(3); 143 assertEquals(2, col.size()); 144 145 146 147 @Test public void test09StressTest() { 148 final int SIZE = 10000; 149 Random randomGenerator = new Random(); 150 151 List<Entry<Integer, String>> entriesList = new LinkedList<>(); // key-Counters: count for every key how many time it was generated 152 153 Map<Integer, Integer> keyCounters = new HashMap<>(); 154 // fill the Tree for (int i = 0; i < SIZE; i++) { 155 int key = (int) (randomGenerator.nextFloat() * SIZE / 3); 156 157 Integer numberOfKeys = keyCounters.get(key); if (numberOfKeys == null) { 158 numberOfKeys = 1; } else { 160 numberOfKeys++; 161 162 keyCounters.put(key, numberOfKeys); 163 Entry<Integer, String> entry = bst.insert(key, "String_" + i); 164 entriesList.add(entry); 165 assertEquals(i + 1, bst.size()); 166 167 168 // verify the number of entries per key for (Map.Entry<Integer, Integer> keyEntry : keyCounters.entrySet()) { 169 170 int key = keyEntry.getKey(); int numberOfKeys = keyEntry.getValue(); 171 assertEquals(numberOfKeys, bst.findAll(key).size()); 173

```
BinarySearchTreeJUnitTest.java
24.9.2018 12:08:35
                                                                                 Page 4/4
175
        // remove all entries
        int size = bst.size();
176
177
        for (Entry<Integer, String> entry : entriesList)
         Entry<Integer, String> deletedEntry = bst.remove(entry);
178
         assertSame(entry, deletedEntry);
         assertEquals(--size, bst.size());
180
181
182
183
184
185
```