```java
/*
 * HSR – Uebungen 'Algorithmen & Datenstrukturen 2'
 * Version: Sun Sep 29 16:37:03 CEST 2019
 */

package uebung03.as.aufgabe03;

import java.util.Collection;

import uebung02.ml.aufgabe01.BinarySearchTree.Entry;


public class AVLTree <K extends Comparable<? super K>, V> {

  private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImpl<K, V>();

  // Show in ADV:
  //private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImplADV<K, V>("AVL-Tree");
  //private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImplADV<K, V>("AVL-Tree", 1, 3)
;
  // Be aware of NodeFixationException!

  public V put(K key, V value) {
    return avlTreeImpl.put(key, value);
  }

  public V get(K key) {
    return avlTreeImpl.get(key);
  }

  public int getHeight() {
    return avlTreeImpl.getHeight();
  }

  public int size() {
    return avlTreeImpl.size();
  }

  public boolean isEmpty() {
    return avlTreeImpl.isEmpty();
  }

  public void clear() {
    avlTreeImpl.clear();
  }

  public Collection<Entry<K, V>> inorder() {
    return avlTreeImpl.inorder();
  }

  public void printInorder() {
    avlTreeImpl.printInorder();
  }

  public void print() {
    avlTreeImpl.print();
  }

  protected AVLTreeImpl<K, V> getImpl() {
    return avlTreeImpl;
  }
```



```java

  public static void main(String[] args) {

    AVLTree<Integer, String> avlTree = new AVLTree<Integer, String>();

    System.out.println("Inserting 2:");
    avlTree.put(2, "Str2");
    avlTree.print();
    System.out.println("================================");
    System.out.println("Inserting 1:");
    avlTree.put(1, "Str1");
    avlTree.print();
    System.out.println("================================");
    System.out.println("Inserting 5:");
    avlTree.put(5, "Str5");
    avlTree.print();
    System.out.println("================================");
    System.out.println("Inserting 3:");
    avlTree.put(3, "Str3");
    avlTree.print();
    System.out.println("================================");
    System.out.println("Inserting 6:");
    avlTree.put(6, "Str6");
    avlTree.print();
    System.out.println("================================");
    System.out.println("Inserting 4:1:");
    avlTree.put(4, "Str4:1");
    avlTree.print();
    System.out.println("================================");
    System.out.println("Inserting 4:2:");
    avlTree.put(4, "Str4:2");
    avlTree.print();
    System.out.println("================================");
    System.out.println("Getting 3 :" + avlTree.get(3));
    System.out.println("Getting 4 :" + avlTree.get(4));
    System.out.println("Getting 7 :" + avlTree.get(7));

  }

}
```

```
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 */
```



```
1  /*
2   * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3   * Version: Sun Sep 29 16:37:03 CEST 2019
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     }
```

```
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
77    } // End of class AVLNode
78
79
80    protected AVLNode getRoot() {
81      return avlNode(root);
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   }
```



```
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  /*
2   * HSR − Uebungen 'Algorithmen & Datenstrukturen 2'
3   * Version: Sun Sep 29 16:37:03 CEST 2019
4   */
5
6  package uebung03.as.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   } // 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) && (maxLeftHeight != −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   }
```



```
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 }
```

```
/*
 * HSR – Uebungen 'Algorithmen & Datenstrukturen 2'
 * Version: Sun Sep 29 16:37:03 CEST 2019
 */

package uebung03.as.aufgabe03;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNull;

import java.util.Collection;
import java.util.LinkedList;

import org.junit.Before;
import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;


@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class AVLTreeJUnitTest {

  AVLTreeImpl<Integer, String> avlTree;

  @Before
  public void setUp() {
    avlTree = new AVLTreeImpl<Integer, String>();
  }

  @Test
  public void test01Put() {
    int[] keys = { 2, 1, 3 };
    String[] expected = {
        " 1 – Str1   : h=0 / parent(key)=2",
        " 2 – Str2   : h=1 ROOT",
        " 3 – Str3   : h=0 \\ parent(key)=2",
    };
    runTest(keys, expected);
    assertEquals(1, avlTree.getHeight());
  }

  @Test
  public void test02Get() {
    int[] keys = { 2, 1, 4, 5, 3 };
    String[] expected = {
        " 1 – Str1   : h=0 / parent(key)=2",
        " 2 – Str2   : h=2 ROOT",
        " 3 – Str3   : h=0 / parent(key)=4",
        " 4 – Str4   : h=1 \\ parent(key)=2",
        " 5 – Str5   : h=0 \\ parent(key)=4",
    };
    runTest(keys, expected);
    assertEquals(2, avlTree.getHeight());
    assertEquals("Str2", avlTree.get(2));
    assertEquals("Str5", avlTree.get(5));
    assertNull(avlTree.get(0));
    assertNull(avlTree.get(6));
  }
```



```

  @Test
  public void test03() {
    int[] keys = { 2, 3, 1 };
    String[] expected = {
        " 1 – Str1   : h=0 / parent(key)=2",
        " 2 – Str2   : h=1 ROOT",
        " 3 – Str3   : h=0 \\ parent(key)=2",
    };
    runTest(keys, expected);
    assertEquals(1, avlTree.getHeight());
    avlTree.put(2, "Str2:2");
    avlTree.put(2, "Str2:3");
    assertEquals(1, avlTree.getHeight());
    expected = new String[] {
        " 1 – Str1   : h=0 / parent(key)=2",
        " 2 – Str2:3 : h=1 ROOT",
        " 3 – Str3   : h=0 \\ parent(key)=2",
    };
    Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
    avlTree.inorder(nodes, avlTree.getRoot());
    verify(nodes, expected);
  }


  private void runTest(int[] keys, String[] expected) {
    for (int key : keys) {
      avlTree.put(key, "Str" + key);
    }
    Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes = new LinkedList<>();
    avlTree.inorder(nodes, avlTree.getRoot());
    assertEquals(expected.length, nodes.size());
    verify(nodes, expected);
  }

  private void verify(Collection<AVLTreeImpl<Integer, String>.AVLNode> nodes, String[]
 expected) {
    int i = 0;
    for (AVLTreeImpl<Integer, String>.AVLNode node: nodes) {
      String nodeStr = node.toString();
      String expectedStr = expected[i];
      assertEquals(expectedStr, nodeStr);
      i++;
    }
  }

}
```