```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Mon Sep 30 16:36:44 CEST 2024
 */

package ex03.baseline.task03;

import java.util.Collection;

import ex02.solution.task01.BinarySearchTree.Entry;


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

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

  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<>();

    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));

  }

}

```

```

/* Session-Log:

Inserting 2:
 2 - Str2   : h=0 ROOT
================================
Inserting 1:
 1 - Str1   : h=0 / parent(key)=2
 2 - Str2   : h=1 ROOT
================================
Inserting 5:
 1 - Str1   : h=0 / parent(key)=2
 2 - Str2   : h=1 ROOT
 5 - Str5   : h=0 \ parent(key)=2
================================
Inserting 3:
 1 - Str1   : h=0 / parent(key)=2
 2 - Str2   : h=2 ROOT
 3 - Str3   : h=0 / parent(key)=5
 5 - Str5   : h=1 \ parent(key)=2
================================
Inserting 6:
 1 - Str1   : h=0 / parent(key)=2
 2 - Str2   : h=2 ROOT
 3 - Str3   : h=0 / parent(key)=5
 5 - Str5   : h=1 \ parent(key)=2
 6 - Str6   : h=0 \ parent(key)=5
================================
Inserting 4:1:
 1 - Str1   : h=0 / parent(key)=2
 2 - Str2   : h=3 ROOT
 3 - Str3   : h=1 / parent(key)=5
 4 - Str4:1 : h=0 \ parent(key)=3
 5 - Str5   : h=2 \ parent(key)=2
 6 - Str6   : h=0 \ parent(key)=5
================================
Inserting 4:2:
 1 - Str1   : h=0 / parent(key)=2
 2 - Str2   : h=3 ROOT
 3 - Str3   : h=1 / parent(key)=5
 4 - Str4:2 : h=0 \ parent(key)=3
 5 - Str5   : h=2 \ parent(key)=2
 6 - Str6   : h=0 \ parent(key)=5
================================
Getting 3 :Str3
Getting 4 :Str4:2
Getting 7 :null

*/
```



```
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Mon Sep 30 16:36:44 CEST 2024
 */

package ex03.baseline.task03;

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

import ex02.solution.task01.BinarySearchTree;

class AVLTreeImpl<K extends Comparable<? super K>, V> extends
    BinarySearchTree<K, V> {

  /**
   * After the BST-operation 'insert()':
   * actionNode shall point to the parent of the new inserted node.
   */
  protected AVLNode actionNode;


  protected class AVLNode extends BinarySearchTree<K, V>.Node {

    private int height;
    private Node parent;

    AVLNode(Entry<K, V> entry) {
      super(entry);
    }

    protected AVLNode setParent(AVLNode parent) {
      AVLNode old = avlNode(this.parent);
      this.parent = parent;
      return old;
    }

    protected AVLNode getParent() {
      return avlNode(parent);
    }

    protected int setHeight(int height) {
      int old = this.height;
      this.height = height;
      return old;
    }

    protected int getHeight() {
      return height;
    }

    @Override
    public AVLNode getLeftChild() {
      return avlNode(super.getLeftChild());
    }

    @Override
    public AVLNode getRightChild() {
      return avlNode(super.getRightChild());
    }
```

```
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 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   }
```



```
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
```

```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Mon Sep 30 16:36:44 CEST 2024
 */

package ex03.baseline.task03;

import ch.hsr.adv.commons.core.logic.domain.styles.ADVStyle;
import ch.hsr.adv.commons.core.logic.util.ADVException;
import ch.hsr.adv.commons.tree.logic.domain.ADVBinaryTreeNode;
import ch.hsr.adv.lib.bootstrapper.ADV;
import ch.hsr.adv.lib.tree.logic.binarytree.BinaryTreeModule;

@SuppressWarnings("unchecked")
public class AVLTreeImplADV<K extends Comparable<? super K>, V>
    extends AVLTreeImpl<K, V> {

  protected BinaryTreeModule advTree;

  protected class AVLNodeADV extends AVLTreeImpl<K, V>.AVLNode
      implements ADVBinaryTreeNode<String> {

    protected AVLNodeADV(Entry<K, V> entry) {
      super(entry);
    }

    @Override
    public String getContent() {
      return getEntry().getKey() + " / " + getEntry().getValue() + " (" + getHeight()
+ ")";
    }

    @Override
    public ADVStyle getStyle() {
      return null;
    }

    @Override
    public AVLNodeADV getLeftChild() {
      return (AVLNodeADV) super.getLeftChild();
    }

    @Override
    public AVLNodeADV getRightChild() {
      return (AVLNodeADV) super.getRightChild();
    }

  } // class AVLTreeImplADV.AVLNodeADV

  public AVLTreeImplADV(String sessionName) {
    this(sessionName, -1, -1);
  }

  public AVLTreeImplADV(String sessionName,
                        int maxLeftHeight, int maxRightHeight) {
    advTree = new BinaryTreeModule(sessionName);
    if ((maxLeftHeight != -1) && (maxLeftHeight != -1)) {
      advTree.setFixedTreeHeight(maxLeftHeight, maxRightHeight);
    }
    try {
      ADV.launch(null);
    } catch (ADVException e) {
      e.printStackTrace();
      System.exit(1);
    }
  }
```



```java

  @Override
  protected Node newNode(Entry<K, V> entry) {
    return new AVLNodeADV(entry);
  }

  @Override
  public V put(K key, V value) {
    V result = super.put(key,  value);
    displayOnADV("put(" + key + "," + value + ")");
    return result;
  }

  protected void displayOnADV(String advMessage) {
    advTree.setRoot((AVLNodeADV) root);
    try {
      ADV.snapshot(advTree, "\n" + advMessage);
    } catch (ADVException e) {
      e.printStackTrace();
      System.exit(2);
    }
  }

}
```

```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Mon Sep 30 16:36:44 CEST 2024
 */

package ex03.baseline.task03;

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;

import ex02.solution.task01.BinarySearchTree;


@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class AVLTreeJUnitTest {

  AVLTreeImpl<Integer, String> avlTree;

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

  @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));
  }
```



```java

  @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<BinarySearchTree<Integer, String>.Node> nodes = new LinkedList<>();
    avlTree.inorder(avlTree.getRoot(), nodes);
    verify(nodes, expected);
  }


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

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

}
```