```java
/*
 * HSR − Uebungen ’Algorithmen & Datenstrukturen 2’
 * Version: Mon Oct  1 20:09:33 CEST 2018
 */

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>();

  // Start the GVS−Server first: Double−Click ’GVS_Server_v1.4.jar’
  //private AVLTreeImpl<K, V> avlTreeImpl = new AVLTreeImplGVS<K, V>();

  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));

    if (avlTree.getImpl() instanceof AVLTreeImplGVS) {
      ((AVLTreeImplGVS<Integer, String>)avlTree.getImpl()).gvsTree.disconnect();
    }

  }

}

```

```
/* 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

*/
```

```
/*
 * HSR − Uebungen 'Algorithmen & Datenstrukturen 2'
 * Version: Mon Oct  1 20:09:33 CEST 2018
 */

package uebung03.as.aufgabe03;

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

import uebung02.ml.aufgabe01.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());
    }
```

```java

    @Override
    public String toString() {
      String result = String.format("%2d - %-6s : h=%d",
                               getEntry().getKey(), getEntry().getValue(), height);
      if (parent == null) {
        result += " ROOT";
      } else {
        boolean left = (parent.getLeftChild() == this) ? true : false;
        result += (left ? " / " : " \\ ") + "parent(key)="
            + parent.getEntry().getKey();
      }
      return result;
    }

  } // End of class AVLNode


  protected AVLNode getRoot() {
    return avlNode(root);
  }

  public V put(K key, V value) {
    // TODO Implement here...
    return null;
  }

  public V get(K key) {
    // TODO Implement here...
    return null;
  }

  @Override
  protected Node insert(Node node, Entry<K, V> entry) {
    // TODO Implement here...
    return null;
  }

  /**
   * The height of the tree.
   *
   * @return The actual height. -1 for an empty tree.
   */
  @Override
  public int getHeight() {
    return height(avlNode(root));
  }

  /**
   * Returns the height of this node.
   *
   * @param node
   * @return The height or -1 if null.
   */
  protected int height(AVLNode node) {
    return (node != null) ? node.getHeight() : -1;
  }

  /**
   * Assures the heights of the tree from 'node' up to the root.
   *
   * @param node
   *          The node from where to start.
   */
  protected void assureHeights(AVLNode node) {
    // TODO Implement here...
  }
```

```java

  /**
   * Assures the correct height for node.
   *
   * @param node
   *          The node to assure its height.
   */
  protected void setHeight(AVLNode node) {
    // TODO Implement here...
  }

  /**
   * Factory-Method. Creates a new node.
   *
   * @param entry
   *          The entry to be inserted in the new node.
   * @return The new created node.
   */
  @Override
  protected Node newNode(Entry<K, V> entry) {
    // TODO Implement here...
    return null;
  }

  /**
   * Generates an inorder-node-list.
   *
   * @param nodeList
   *          The node-list to fill in inorder.
   * @param node
   *          The node to start from.
   */
  protected void inorder(Collection<AVLNode> nodeList, AVLNode node) {
    if (node == null)
      return;
    inorder(nodeList, node.getLeftChild());
    nodeList.add(node);
    inorder(nodeList, node.getRightChild());
  }

  // Type-Casting: Node -> AVLNode (Cast-Encapsulation)
  @SuppressWarnings("unchecked")
  protected AVLNode avlNode(Node node) {
    return (AVLNode)node;
  }

  public void print() {
    List<AVLNode> nodeList = new LinkedList<>();
    inorder(nodeList, avlNode(root));
    for (AVLNode node: nodeList) {
      System.out.println(node + "  ");
    }
  }

}


```

```java
/*
 * HSR − Uebungen ’Algorithmen & Datenstrukturen 2’
 * Version: Mon Oct  1 20:09:33 CEST 2018
 */

package uebung03.as.aufgabe03;

import gvs.tree.GVSBinaryTreeNode;
import gvs.tree.GVSTreeWithRoot;
import gvs.typ.node.GVSNodeTyp;

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

  protected GVSTreeWithRoot gvsTree;

  private final int DELAY = 200;

  protected class AVLNodeGVS extends AVLTreeImpl<K, V>.AVLNode implements GVSBinaryTre
eNode {

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

    public GVSBinaryTreeNode getGVSLeftChild() {
      return (GVSBinaryTreeNode) getLeftChild();
    }

    public GVSBinaryTreeNode getGVSRightChild() {
      return (GVSBinaryTreeNode) getRightChild();
    }

    public String getNodeLabel() {
      Entry<K, V> e = getEntry();
      return e.getKey() + " "+ e.getValue();
      //return e.getKey().toString();
    }

    public GVSNodeTyp getNodeTyp() {
      return null;
    }

  } // class BinaryTreeTestGVS.NodeGVS


  AVLTreeImplGVS() {
    this("AVLTreeGVS");
  }

  AVLTreeImplGVS(String title) {
    gvsTree = new GVSTreeWithRoot(title);
  }

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

  @Override
  public V put(K key, V value) {
    V result = super.put(key, value);
    gvsTree.setRoot((GVSBinaryTreeNode) root);
    gvsTree.display();
    try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
    return result;
  }

```

```java

  @Override
  public Entry<K, V> insert(K key, V value) {
    Entry<K, V> newEntry = super.insert(key, value);
    gvsTree.setRoot((GVSBinaryTreeNode) root);
    gvsTree.display();
    try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
    return newEntry;
  }

  @Override
  public Entry<K, V> remove(Entry<K, V> entry) {
    Entry<K, V> deletedEntry = super.remove(entry);
    gvsTree.display();
    try {Thread.sleep(DELAY);} catch (InterruptedException e) {}
    return deletedEntry;
  }

}



```

```java
/*
 * HSR − Uebungen 'Algorithmen & Datenstrukturen 2'
 * Version: Mon Oct  1 20:09:33 CEST 2018
 */

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.After;
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() {
    //System.setProperty("NoGVS", "true");
    avlTree = new AVLTree<Integer, String>().getImpl();
  }

  @After
  public void tearDown() {
    if (avlTree instanceof AVLTreeImplGVS) {
      ((AVLTreeImplGVS<Integer, String>)avlTree).gvsTree.disconnect();
    }
  }

  @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<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++;
    }
  }

}
```