**package** problem1;

**import** java.util.Iterator;

**import** java.util.TreeMap;

**import** java.util.TreeSet;

**public** **class** MyBST {

/\*\* The tree root. \*/

**private** BinaryNode root;

**public** MyBST() {

root = **null**;

}

**public** **void** preOrder() {

preOrder(root);

}

**private** **void** preOrder(BinaryNode t) {

**if** (t != **null**) {

**int** value\_root = t.element;

System.***out***.printf("%s -> ", value\_root);

preOrder(t.left);

preOrder(t.right);

}

}

**public** **void** postOrder() {

postOrder(**this**.root);

}

**private** **void** postOrder(BinaryNode t) {

**if** (t != **null**) {

postOrder(t.left);

postOrder(t.right);

**int** value\_root = t.element;

System.***out***.printf("%s -> ", value\_root);

}

}

**public** **boolean** contains(Integer key) {

**if** (**this**.root == **null**) {

**return** **false**;

}

**return** contains(key, **this**.root);

}

**public** Integer getRoot() {

**return** **this**.root.element;

}

**public** Integer leafNodes() {

**return** leafNodes(**this**.root);

}

**private** **int** leafNodes(BinaryNode t) {

**if** (t == **null**) {

**return** -1;

}

**return** t.left.element;

}

**public** **int** size() {

**if** (**this**.root == **null**) {

**return** 0;

}

**return** size(**this**.root);

}

**public** **int** size(BinaryNode tree) {

**if** (tree == **null**) {

**return** 0;

}

**return** 1 + size(tree.left) + size(tree.right);

}

**public** **boolean** isEmpty() {

**return** **this**.root == **null**;

}

**public** Integer findMin() {

**if** (**this**.root == **null**) {

**return** -1;

}

**return** findMin(**this**.root, **this**.root.element);

}

**private** Integer findMin(BinaryNode tree, **int** key) {

**if** (tree == **null**) {

**return** key;

}

**if** (tree.element < key) {

key=tree.element;

}

**int** value\_left = findMin(tree.left, key);

**int** value\_right = findMin(tree.right, key);

**if** (value\_left < key) {

key = value\_left;

}

**if** (value\_right < key) {

key = value\_right;

}

**return** key;

}

**public** Integer findMax() {

**if** (**this**.root == **null**) {

**return** -1;

}

**return** findMax(**this**.root, **this**.root.element);

}

**private** Integer findMax(BinaryNode tree, **int** key) {

**if** (tree == **null**) {

**return** key;

}

**if** (tree.element > key) {

key= tree.element;

}

**int** value\_left = findMax(tree.left, key);

**int** value\_right = findMax(tree.right, key);

**if** (value\_left > key) {

key = value\_left;

}

**if** (value\_right > key) {

key = value\_right;

}

**return** key;

}

**public** **boolean** contains(Integer key, BinaryNode tree) {

**if** (tree != **null**) {

**int** value\_root = tree.element;

**if** (value\_root == key) {

**return** **true**;

}

**return** contains(key, tree.left) || contains(key, tree.right);

}

**return** **false**;

}

/\*\*

\* Prints the values in the nodes of the tree in sorted order. Inorder Traversal

\*/

**public** **void** printTree() {

**if** (root == **null**)

System.***out***.println("Empty tree");

**else**

printTree(root);

}

// Inorder Traversal to print the nodes in Ascending order

**private** **void** printTree(BinaryNode t) {

**if** (t != **null**) {

printTree(t.left);

System.***out***.print(t.element + ",");

printTree(t.right);

}

}

// Assume the data in the Node is an Integer.

**public** **void** insert(Integer x) {

**if** (root == **null**) {

root = **new** BinaryNode(x);

**return**;

} **else** {

BinaryNode n = root;

**boolean** inserted = **false**;

**while** (!inserted)// true

{

**if** (x.compareTo(n.element) < 0) {

// space found on the left

**if** (n.left == **null**) {

n.left = **new** BinaryNode(x, **null**, **null**);

inserted = **true**;

}

// keep looking for a place to insert (a null)

**else** {

n = n.left;

}

} **else** **if** (x.compareTo(n.element) > 0) {

// space found on the right

**if** (n.right == **null**) {

n.right = **new** BinaryNode(x, **null**, **null**);

inserted = **true**;

}

// keep looking for a place to insert (a null)

**else** {

n = n.right;

}

}

// if a node already exists

**else** {

inserted = **true**;

}

}

}

}

**private** **class** BinaryNode {

**private** Integer element;// The data in the node

**private** BinaryNode left; // Left child

**private** BinaryNode right; // Right child

// Constructors

BinaryNode(Integer theElement) {

**this**(theElement, **null**, **null**);

}

BinaryNode(Integer element, BinaryNode left, BinaryNode right) {

**this**.element = element;

**this**.left = left;

**this**.right = right;

}

}

**public** **static** **void** main(String[] args) {

MyBST mybst = **new** MyBST();

**int**[] a = { 15, 12, 9, 56, 1, 16, 19, 22, 3, 100, 2, 25,-9999 };

**for** (**int** j = 0; j < a.length; j++) {

mybst.insert(a[j]);

}

mybst.insert(12);

System.***out***.print("\n Our Tree");

mybst.printTree();

System.***out***.print("\n Part A Pre ordwer");

mybst.preOrder();

System.***out***.print("\n Part B Post Order");

mybst.postOrder();

System.***out***.printf("\n Part C , We will check if the mybst has -9999 or not and result ot it is %s", mybst.contains(-9999));

System.***out***.printf("\n Part D , We will check the root of mybst and the value is %s", mybst.getRoot());

System.***out***.printf("\n Part E , We will leafNodes the mybst and the value is %s", mybst.leafNodes());

System.***out***.printf("\n Part F , We will size the mybst and the value is %s", mybst.size());

System.***out***.printf("\n Part G , We if the mybst is empty and the answer is %s", mybst.isEmpty());

System.***out***.printf("\n Part H , We the min value of mybst and the answer is is %s", mybst.findMin());

System.***out***.printf("\n Part i , We the max value of mybst and the answer is is %s", mybst.findMax());

// TreeSet<Integer> ts = new TreeSet<Integer>();

//

// for (int j = 0; j < a.length; j++) {

// ts.add(a[j]);

// System.out.println("\nAfter inserting " + j + "th item " + a[j]);

// Iterator<Integer> it = ts.iterator();

// Integer nextItem = null;

// while (it.hasNext()) {

// nextItem = it.next();

// System.out.print(nextItem + " ");

// }

// System.out.println();

// }

//

// TreeMap<Integer, String> map = new TreeMap<Integer, String>();

// map.put(8, "Hello");

// map.put(10, "World!");

// map.put(11, "Welcome");

// map.remove(8);

// String str = map.get(11) + ", " + map.get(10);

// System.out.println(str);

}

}

