**public** **class** BST<Key **extends** Comparable<Key>, Value> {

Node root; // root of the tree

**public** **boolean** isEmpty() {   
 **return** size() == 0;   
 }

**class** Node {

Key key;

Value val;

Node left, right; // subtrees

**int** N;

**public** Node(Key key, Value val, **int** N) {

**this**.key = key;

**this**.val = val;

**this**.N = N;

}

}

/\*\* Return # key-value pairs. \*/

**public** **int** size() {   
 **return** size(root);   
 }

// Helper method deals with "empty nodes"

**private** **int** size(Node node) {

**if** (node == **null**) **return** 0;

**return** node.N;

}

**public** **boolean** contains(Key key) { **return** get(key) != **null**; }

/\*\* Search parent. \*/

**public** Value get(Key key) { **return** get(root, key); }

**private** Value get(Node parent, Key key) {

**if** (parent == **null**) **return** **null**;

**int** cmp = key.compareTo(parent.key);

**if** (cmp < 0) **return** get(parent.left, key);

**else** **if** (cmp > 0) **return** get(parent.right, key);

**else** **return** parent.val;

}

/\*\* Invoke put on parent, should it exist. \*/

**public** **void** put(Key key, Value val) {

root = put(root, key, val);

}

**private** Node put(Node parent, Key key, Value val) {

**if** (parent == **null**) **return** **new** Node(key, val, 1);

**int** cmp = key.compareTo(parent.key);

**if** (cmp < 0) parent.left = put(parent.left, key, val);

**else** **if** (cmp > 0) parent.right = put(parent.right, key, val);

**else** parent.val = val;

parent.N = 1 + size(parent.left) + size(parent.right);

**return** parent;

}

}