**Main**

**public** **class** BinarySearchTreeTest {

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

// **TODO** Auto-generated method stub

BinarySearchTree T = **new** BinarySearchTree();

T.insert("S");

T.insert("J");

T.insert("U");

T.insert("B");

T.insert("M");

T.insert("A");

T.insert("D");

T.insert("R");

T.insert("G");

T.insert("Q");

T.insert("E");

System.***out***.println("이트리는");

T.printBST();

System.***out***.println("로 구성되어 있습니다.");

System.***out***.println();

String key = "A";

TreeNode P = T.find(key);

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

System.***out***.println("찾으시는 값" + key + "를 찾았습니다.");

} **else** {

System.***out***.println("찾으시는 값" + key + "를 못찾았습니다.");

}

System.***out***.println();

String key1 = "G";

System.***out***.println(key1 + "값을 삭제합니다.");

T.printBST();

P = T.find(key1);

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

T.delete(key1);

} **else** {

System.***out***.println("삭제하려는 값" + key1 + "를 못찾앗습니다.");

}

T.printBST();

System.***out***.println("다시 값을 검색합니다.");

P = T.find(key1);

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

System.***out***.println("찾으시는 값" + key1 + "를 찾았습니다.");

} **else** {

System.***out***.println("찾으시는 값" + key1 + "를 못찾았습니다.");

}

System.***out***.println();

BinarySearchTree cBST = **new** BinarySearchTree();

BinarySearchTree bBST = **new** BinarySearchTree();

String spli = "J";

T.spliBST(spli, bBST, cBST);

System.***out***.println(spli + "로 분할한 트리 1");

bBST.printBST();

System.***out***.println(spli+ "로 분할한 트리 2");

cBST.printBST();

}

}

BinarySearchTree

**class** BinarySearchTree {

**private** TreeNode rootNode;

**private** TreeNode insertKey(TreeNode T, String x) {

**if** (T == **null**) {

TreeNode newNode = **new** TreeNode();

newNode.key = x;

**return** newNode;

} **else** **if** (x.compareTo(T.key) < 0) {

T.left = insertKey(T.left, x);

**return** T;

} **else** **if** (x.compareTo(T.key) > 0) {

T.right = insertKey(T.right, x);

**return** T;

} **else** {

**return** T;

}

}

**void** insert(String x) {

rootNode = insertKey(rootNode, x);

}

**private** **void** printNode(TreeNode N) {

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

System.***out***.print("(");

printNode(N.left);

System.***out***.print(N.key);

printNode(N.right);

System.***out***.print(")");

}

}

**void** printBST() {

printNode(rootNode);

System.***out***.println();

}

TreeNode find(String x) {

TreeNode T = rootNode;

**int** result;

**while** (T != **null**) {

**if** ((result = x.compareTo(T.key)) < 0) {

T = T.left;

} **else** **if** (result == 0) {

**return** T;

} **else** {

T = T.right;

}

}

**return** T;

}

TreeNode maxNode(TreeNode B) {

TreeNode p;

p = B;

**if** (p == **null**) {

**return** p;

} **else** **if** (p.right == **null**)

**return** p;

**else** {

**while** (p.right != **null**) {

p = p.right;

}

**return** p;

}

}

**void** delete(String x) {

rootNode = deleteKey(rootNode, x);

}

TreeNode deleteKey(TreeNode T, String x) {

TreeNode p;

TreeNode parent;

TreeNode q;

p = T;

parent = **null**;

**while** (p != **null** && p.key != x) {

parent = p;

**if** ((x.compareTo(p.key)) < 0) {

p = p.left;

} **else** {

p = p.right;

}

}

**if** (p == **null**)

**return** **null**;

**if** (p.left == **null** && p.right == **null**) {

**if** (parent.left == p) {

parent.left = **null**;

} **else**

parent.right = **null**;

}

**else** **if** (p.left == **null** || p.right == **null**) {

**if** (p.left != **null**) {

**if** (parent.left == p) {

parent.left = p.left;

} **else** {

parent.right = p.left;

}

} **else** {

**if** (parent.left == p) {

parent.left = p.right;

} **else** {

parent.right = p.right;

}

}

} **else** **if** (p.left != **null** && p.right != **null**) {

q = maxNode(p.left);

p.key = q.key;

deleteKey(p.left, p.key);

}

**return** T;

}

**boolean** spliBST(String x, BinarySearchTree bBST, BinarySearchTree cBST) {

TreeNode Small = **new** TreeNode();

TreeNode Large = **new** TreeNode();

TreeNode S = Small;

TreeNode L = Large;

TreeNode P = rootNode;

**while** (P != **null**) {

**if** (x == P.key) {

S.right = P.left;

L.left = P.right;

Small = Small.right;

Large = Large.left;

bBST.rootNode = Small;

cBST.rootNode = Large;

**return** **true**;

} **else** **if** (x.compareTo(P.key) < 0) {

L.left = P;

L = P;

P = P.left;

} **else** {

S.right = P;

S = P;

P = P.right;

}

}

S.right = **null**;

L.left = **null**;

Small = Small.right;

Large = Large.left;

bBST.rootNode = Small;

cBST.rootNode = Large;

**return** **false**;

}

}

TreeNode

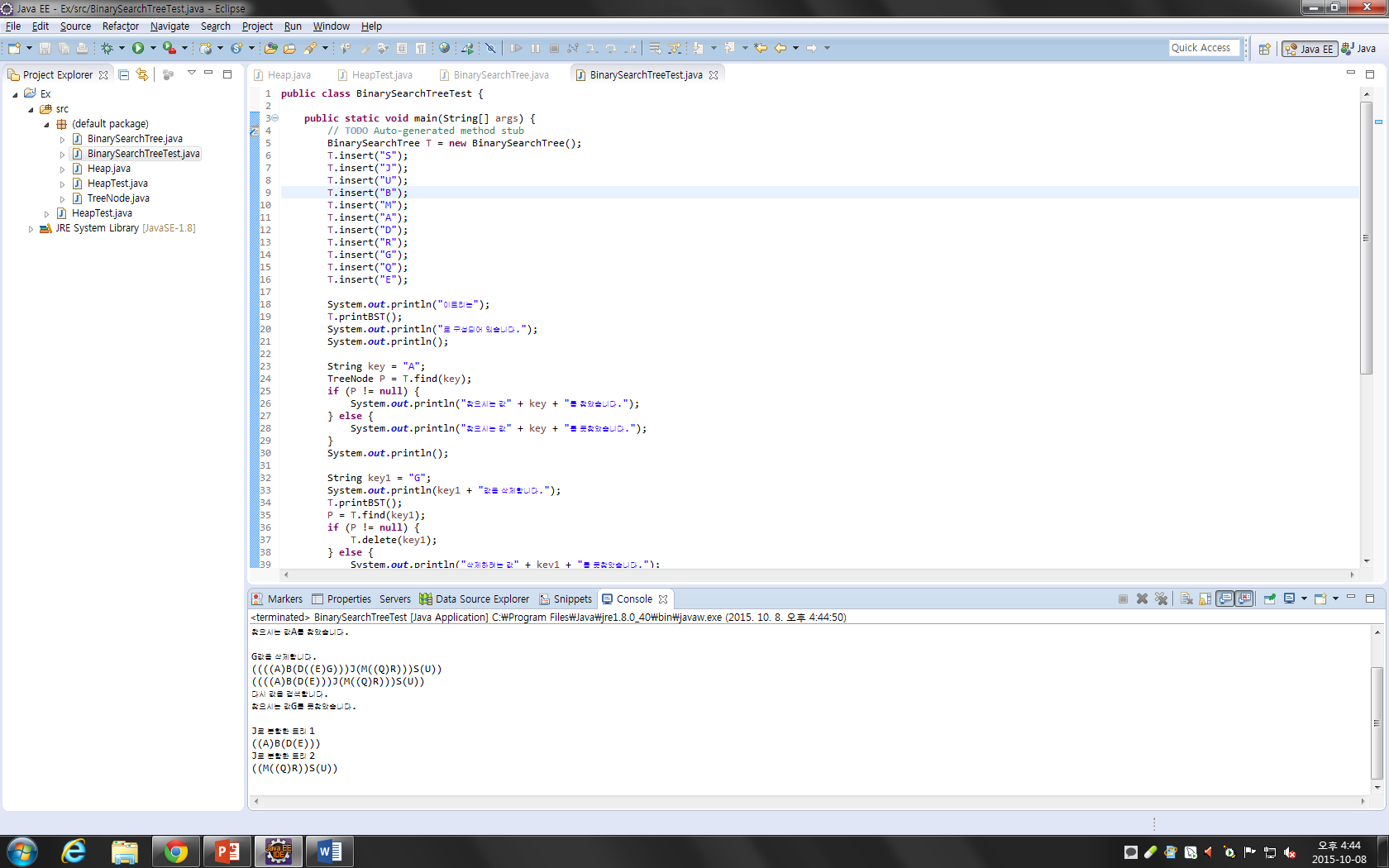
**public** **class** TreeNode {

String key;

TreeNode left;

TreeNode right;

}



Main

**public** **class** HeapTest {

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

// **TODO** Auto-generated method stub

Heap hp = **new** Heap();

hp.insertHeap(18);

hp.insertHeap(13);

hp.insertHeap(5);

hp.insertHeap(12);

hp.insertHeap(8);

hp.insertHeap(19);

System.***out***.print("입력 된 값: ");

hp.printHeap();

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

}

}

Heap

**public** **class** Heap {

**int** maxSize;

**int** n;

**int** heap[];

Heap(){

maxSize = 50;

n = 0;

heap = **new** **int**[maxSize];

}

**void** insertHeap(**int** item){

**int** i;

**if**(n == maxSize)

System.***out***.println("heapFull!");

n=n+1;

**for**(i = n;;){

**if**(i == 1) **break**;

**if**(item <= heap[i/2]) **break**;

heap[i] = heap[i/2];

i=i/2;

}

heap[i] = item;

}

**public** **void** printHeap(){

**for**(**int** i = 1;i<=n;i++){

System.***out***.print(heap[i]+" ");

}

System.***out***.println();

}

}

