A

B,t C

D - E,t F

G H I J - K

A

C

D - F

G H I J - K

A, D, I, J

Class Node {

Char data,

Node left;

Node right;

Boolean deletable;

}

Class TreeDeletion {

List<Node> deleteNode( Node root) {

List<Node> rootNodes = new ArrayList<>();

if(root == null) {

Return rootNodes;

}

if(root.deletable == false) {

rootNodes.add(root);

}

Queue<Node> currLevel = new Queue<Node>();

currLevel.add(root);

while(currLevel.size() >0) {

Queue<Node> nextLevel = new Queue<Node>();

for( Node node : currLevel ) {

// Level Order Next Level

if(node.left != null) {

nextLevel.add(node.left);

}

if(node.right != null) {

nextLevel.add(node.right);

}

//

if(node.deletable == true) {

if(node.left != null && node.left.deletable != true) {

rootNodes.add(node.left);

}

if(node.right != null && node.right.deletable != true) {

rootNodes.add(node.right);

}

node = null;

}

// Delete the deletable node.

if(node.left != null && node.left.deletable == true) {

Node.left = null;

}

if(node.right != null && node.right.deletable == true) {

Node.right = null;

}

}

currLevel = nextLevel;

}

}

main () {

Node a = new Node(“A”);

Node b = new Node(“B”);

a.left = b;

b = null;

if(a.left == null)

}

}

Class Singleton {

Private static Singleton instance = null;

ReadWriteLock rwLock = new ReadWriteLock();

Public static Singleton getInstance() {

if(instance == null)

rwLock.lock();

if(instance == null)

instance = new Singleton();

rwLock.unlock();

}

Return instance;

}

}