Data Structure

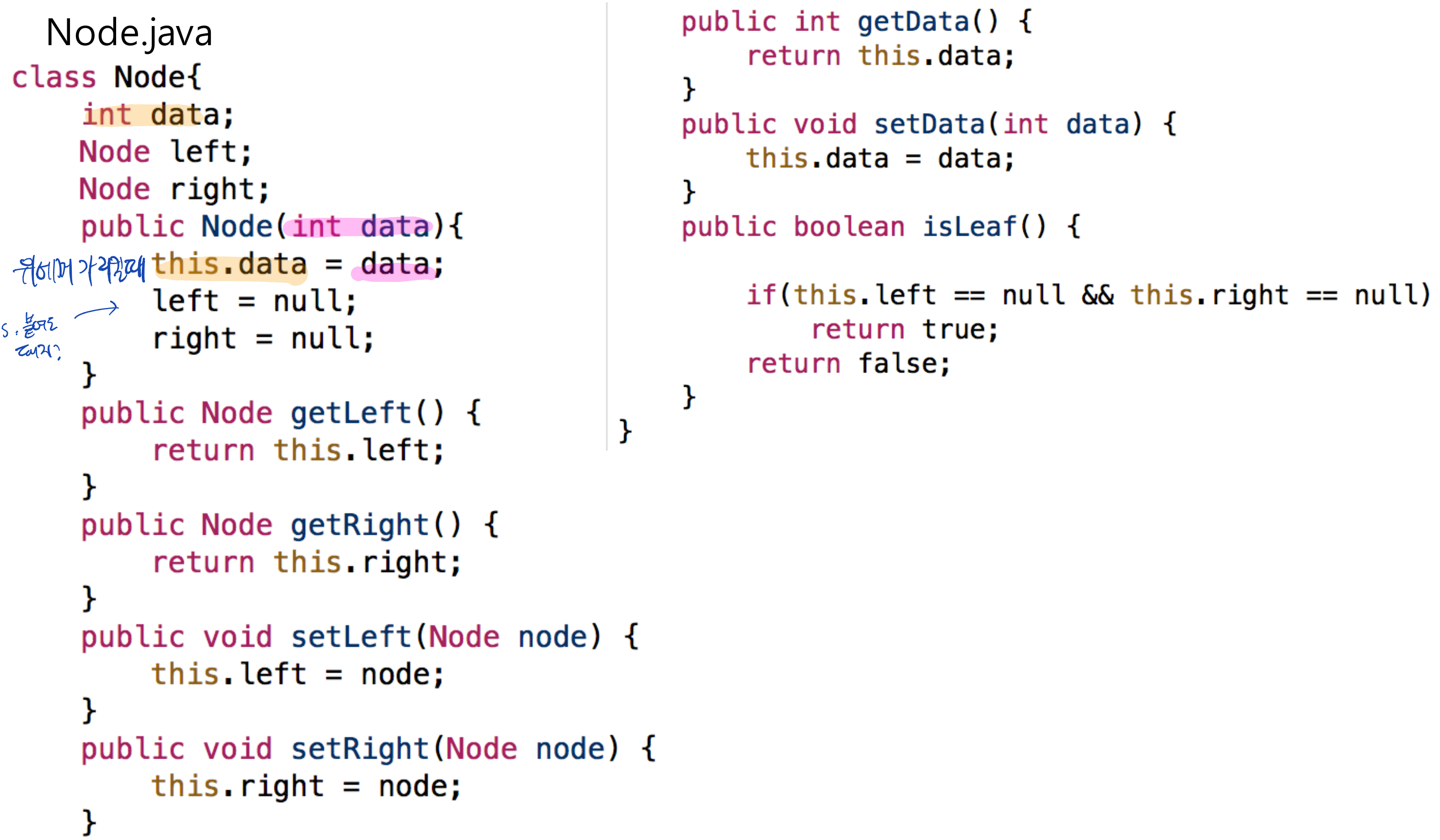
(Java programming)

Chapter 12 answer.

KONKUK UNIVERSITY KOREA
1946
건국대학교

• Binary Search Trees

Node.java



위에꺼 가리킬때

this. 붙여도 돼지? →

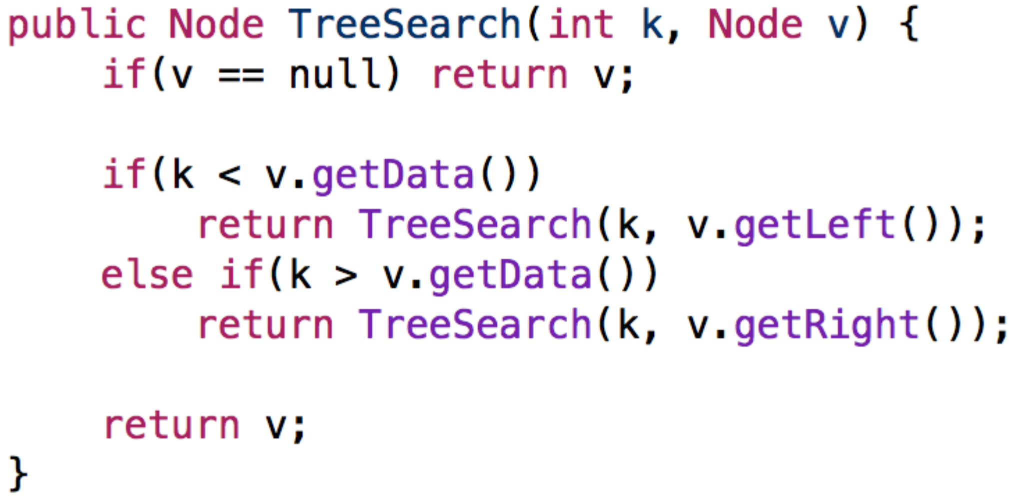
```java
class Node{
    int data;
    Node left;
    Node right;
    public Node(int data){
        this.data = data;
        left = null;
        right = null;
    }
    public Node getLeft() {
        return this.left;
    }
    public Node getRight() {
        return this.right;
    }
    public void setLeft(Node node) {
        this.left = node;
    }
    public void setRight(Node node) {
        this.right = node;
    }
    public int getData() {
        return this.data;
    }
    public void setData(int data) {
        this.data = data;
    }
    public boolean isLeaf() {

        if(this.left == null && this.right == null)
            return true;
        return false;
    }
}
```

• Binary Search Trees

TreeSearch

data

node



```
public Node TreeSearch(int k, Node v) {
    if(v == null) return v;

    if(k < v.getData())
        return TreeSearch(k, v.getLeft());
    else if(k > v.getData())
        return TreeSearch(k, v.getRight());

    return v;
}
```

까보기

리커시브 - 지가 다 해줄 거라고 생각하기

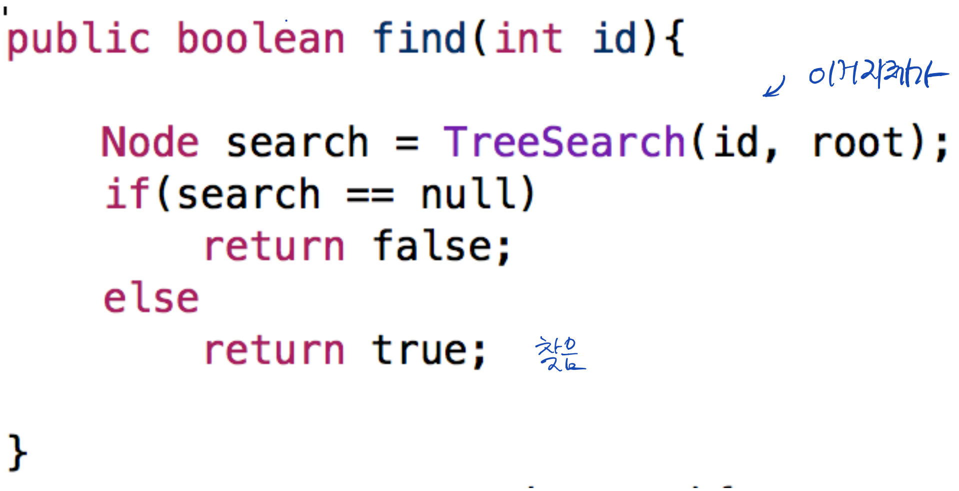
• Binary Search Trees

Find

null 이면 false

null이 아니면 데이터가 있다는거니까 true.

ex) 82.



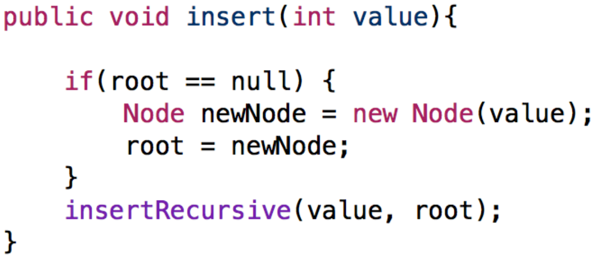
```
public boolean find(int id){

    Node search = TreeSearch(id, root);
    if(search == null)
        return false;
    else
        return true;

}
```

이거자체가 Treesearch 실행.

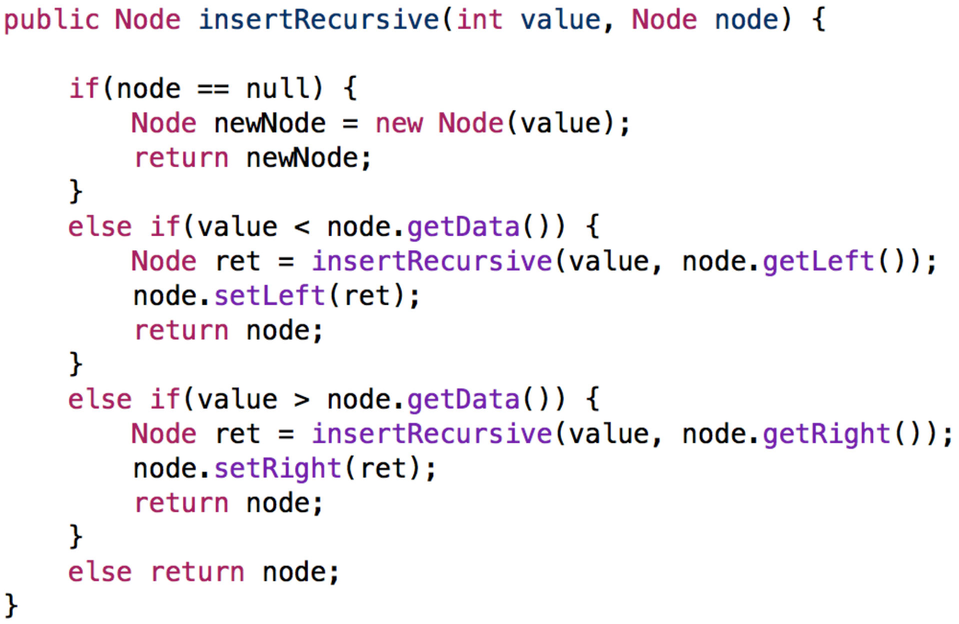
찾음

• Binary Search Trees

Insert



```
public void insert(int value){

    if(root == null) {
        Node newNode = new Node(value);
        root = newNode;
    }
    insertRecursive(value, root);
}
```



```
public Node insertRecursive(int value, Node node) {

    if(node == null) {
        Node newNode = new Node(value);
        return newNode;
    }
    else if(value < node.getData()) {
        Node ret = insertRecursive(value, node.getLeft());
        node.setLeft(ret);
        return node;
    }
    else if(value > node.getData()) {
        Node ret = insertRecursive(value, node.getRight());
        node.setRight(ret);
        return node;
    }
    else return node;
}
```

• Binary Search Trees

Delete

```
public boolean delete(int value){
    Node parent = root;
    Node current = root;
    boolean isLeftChild = false;
    while(current.getData()!=value){
        parent = current;
        if(current.getData()>value){
            isLeftChild = true;
            current = current.getLeft();
        }else{
            isLeftChild = false;
            current = current.getRight();
        }
        if(current ==null){
            return false;
        }
    }
```

해당노드찾음 //

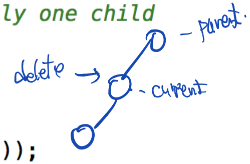
(find 쓰면 왜 안돼지?)

```
//if i am here that means we have found the node
//Case 1: if node to be deleted has no children
if(current.getLeft()==null && current.getRight()==null){
    if(current==root){
        root = null;
    }
    if(isLeftChild ==true){
        parent.setLeft(null);
    }else{
        parent.setRight(null);
    }
}
//Case 2 : if node to be deleted has only one child
else if(current.getRight()==null){
    if(current==root){
        root = current.getLeft();
    }else if(isLeftChild){
        parent.setLeft(current.getLeft());
    }else{
        parent.setRight(current.getLeft());
    }
}
else if(current.getLeft()==null){
    if(current==root){
        root = current.getRight();
    }else if(isLeftChild){
        parent.setLeft(current.getRight());
    }else{
        parent.setRight(current.getRight());
    }
}
```

↑ 둘다 자식이 없는 경우

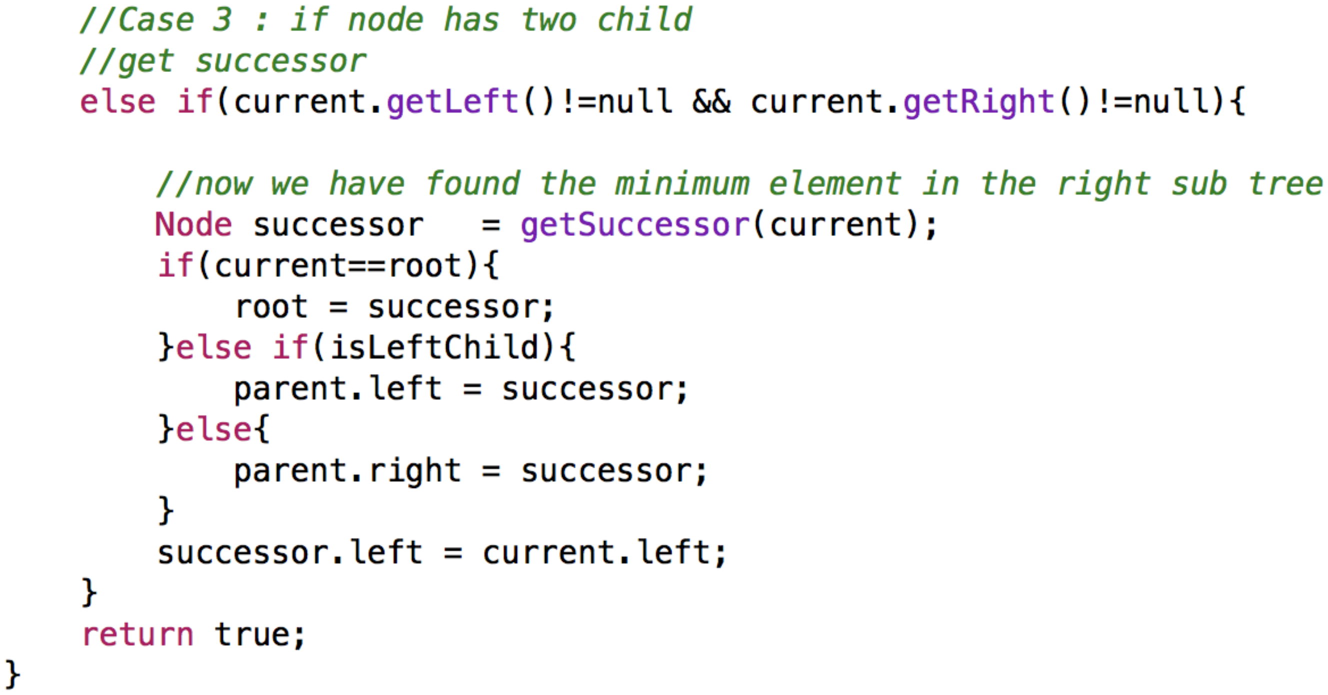
이과정이 왜 필요하지?

left 있으면 null로 바꿈.



• Binary Search Trees

Delete

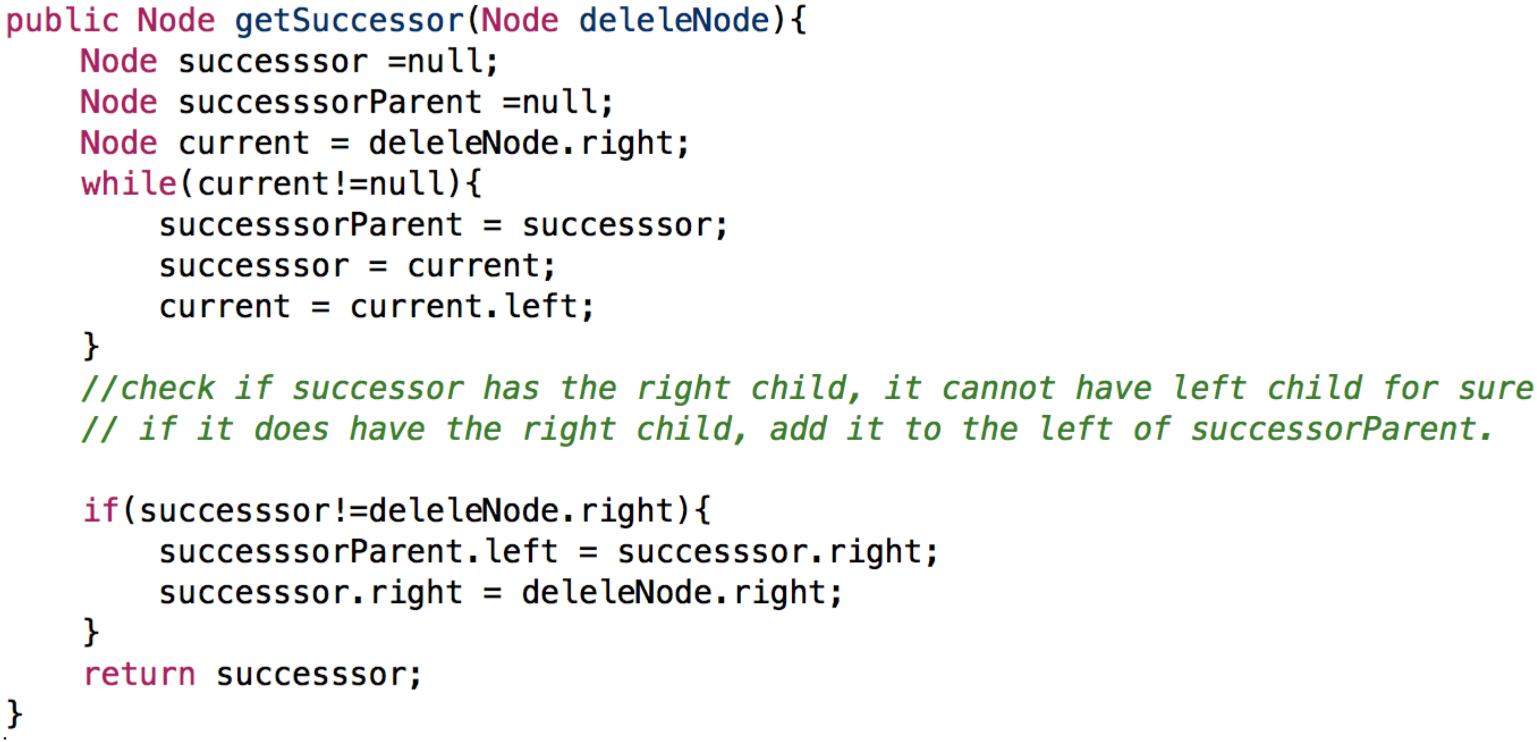


```
    //Case 3 : if node has two child
    //get successor
    else if(current.getLeft()!=null && current.getRight()!=null){

        //now we have found the minimum element in the right sub tree
        Node successor   = getSuccessor(current);
        if(current==root){
            root = successor;
        }else if(isLeftChild){
            parent.left = successor;
        }else{
            parent.right = successor;
        }
        successor.left = current.left;
    }
    return true;
}
```

• Binary Search Trees

Delete

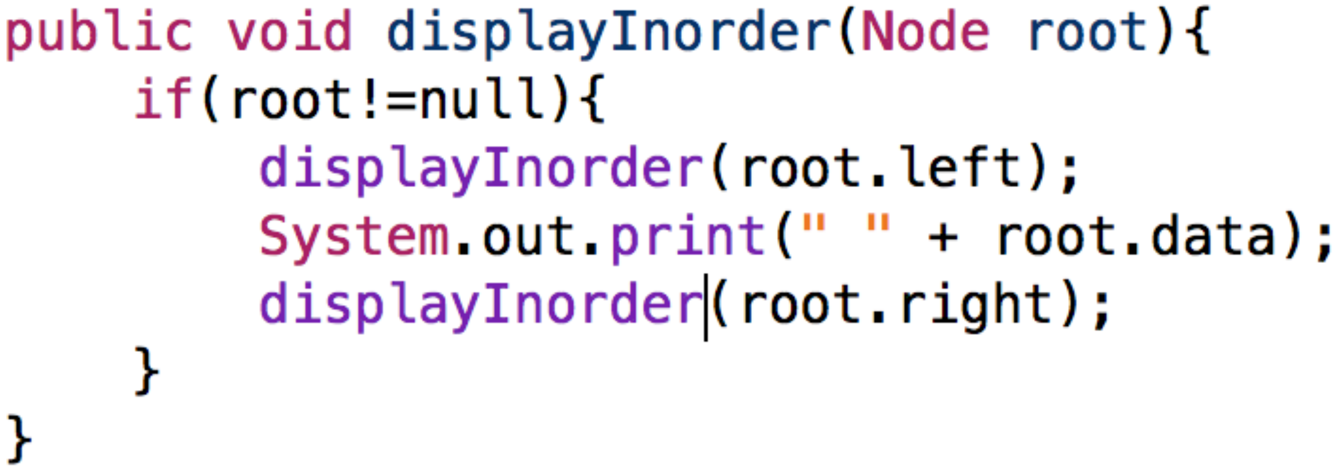
(to get minimum node in the right subtree of deletion node)



```
public Node getSuccessor(Node deleleNode){
    Node successsor =null;
    Node successsorParent =null;
    Node current = deleleNode.right;
    while(current!=null){
        successsorParent = successsor;
        successsor = current;
        current = current.left;
    }
    //check if successor has the right child, it cannot have left child for sure
    // if it does have the right child, add it to the left of successorParent.

    if(successsor!=deleleNode.right){
        successsorParent.left = successsor.right;
        successsor.right = deleleNode.right;
    }
    return successsor;
}
```

- ***Binary Search Trees***

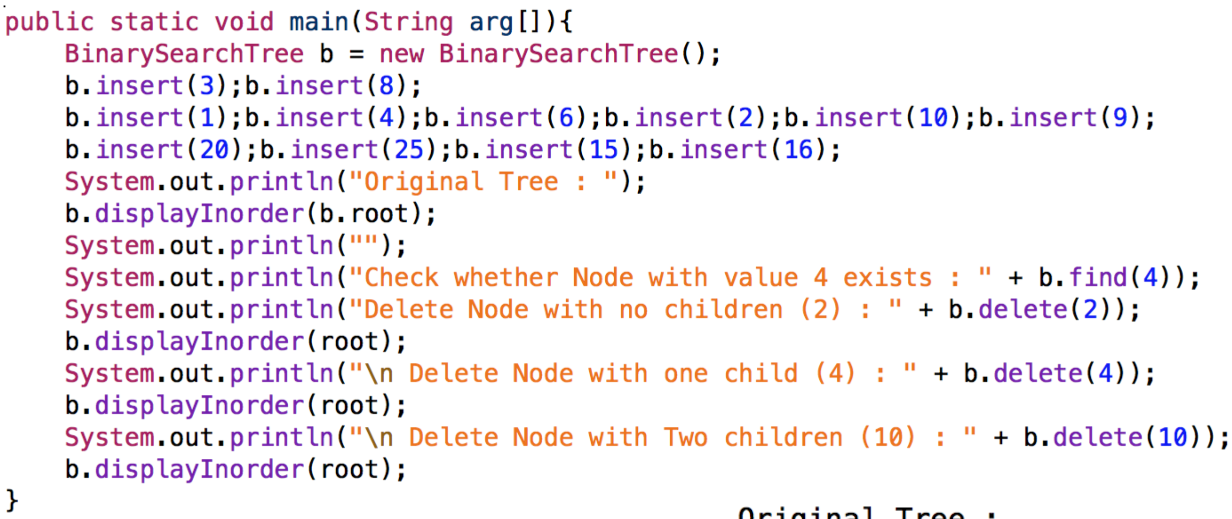
Display inorder



```
public void displayInorder(Node root){
    if(root!=null){
        displayInorder(root.left);
        System.out.print(" " + root.data);
        displayInorder(root.right);
    }
}
```

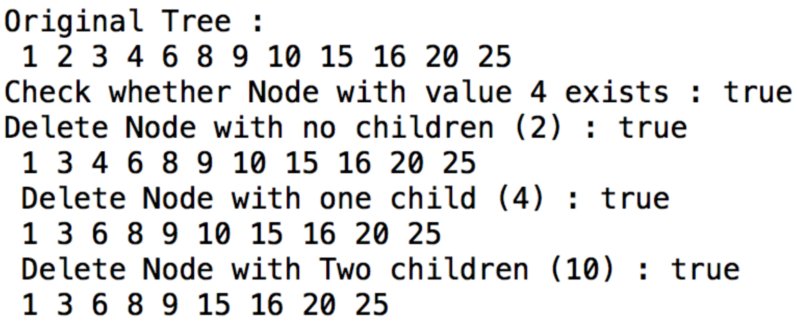
• Binary Search Trees

main



```
public static void main(String arg[]){
    BinarySearchTree b = new BinarySearchTree();
    b.insert(3);b.insert(8);
    b.insert(1);b.insert(4);b.insert(6);b.insert(2);b.insert(10);b.insert(9);
    b.insert(20);b.insert(25);b.insert(15);b.insert(16);
    System.out.println("Original Tree : ");
    b.displayInorder(b.root);
    System.out.println("");
    System.out.println("Check whether Node with value 4 exists : " + b.find(4));
    System.out.println("Delete Node with no children (2) : " + b.delete(2));
    b.displayInorder(root);
    System.out.println("\n Delete Node with one child (4) : " + b.delete(4));
    b.displayInorder(root);
    System.out.println("\n Delete Node with Two children (10) : " + b.delete(10));
    b.displayInorder(root);
}
```

Output:



```
Original Tree :
 1 2 3 4 6 8 9 10 15 16 20 25
Check whether Node with value 4 exists : true
Delete Node with no children (2) : true
 1 3 4 6 8 9 10 15 16 20 25
 Delete Node with one child (4) : true
 1 3 6 8 9 10 15 16 20 25
 Delete Node with Two children (10) : true
 1 3 6 8 9 15 16 20 25
```