```
package datastructure.Tree;

class BinarySearchTree {
    Node root;
    BinarySearchTree() {
        root = null;
    }

    /* Let us create following BST
          8
       /     \
      4       12
     /  \    /  \
    1   7   9    14 */

    void insertNode(int key) {
        root = insertHelper(root, key);
    }

    Node insertHelper(Node root, int key) {
        if (root == null) {
            root = new Node(key);
            return root;
        }
        if (key < root.key)
            root.left = insertHelper(root.left, key);
        else if (key > root.key)
                root.right = insertHelper(root.right, key);
```

```
    return root;
}

void preOrder() {
    preOrderHelper(root);
}

void preOrderHelper(Node root) {
    if (root != null) {
        System.out.println(root.key);
        preOrderHelper(root.left);
        preOrderHelper(root.right);
    }
}

void inorder() {
    inorderHelper(root);
}

void inorderHelper(Node root) {
    if (root != null) {
        inorderHelper(root.left);
        System.out.println(root.key);
        inorderHelper(root.right);
    }
}
```

```
void postOrder() {
    postOrderHelper(root);
}

void postOrderHelper(Node root) {
    if (root != null) {
        postOrderHelper(root.left);
        postOrderHelper(root.right);
        System.out.println(root.key);
    }
}


void deleteKey(int key) {
    root = deleteNode(root, key);
}
/*
      8
   /     \
  4       14
 /  \    /  \
1   7   9    */

Node deleteNode(Node root, int key) {
    if (root == null)
        return root;

    if (key < root.key) {
```

```
            root.left = deleteNode(root.left, key);
        } else if (key > root.key) {
            root.right = deleteNode(root.right, key);
        } else {
            // node with no leaf nodes
            if (root.left == null && root.right == null) {
                return null;
            } else if (root.left == null) {
                // node with one node (no left node)
                return root.right;
            } else if (root.right == null) {
                // node with one node (no right node)
                return root.left;
            } else {
                // nodes with two nodes
                // search for min number in right sub tree
                int minValue = minValue(root.right);
                root.key = minValue;
                root.right = deleteNode(root.right, minValue);
            }
        }

        return root;
    }

    int minValue(Node root) {
        int minv = root.key;
        while (root.left != null) {
```

```
        minv = root.left.key;
        root = root.left;
    }
    return minv;
}

/*
      8
   /     \
  4       12
 /  \    /  \
1    7  9   14 */

public Node search(Node root, int key) {
    if (root == null || root.key == key)
        return root;

    if (key < root.key)
        return search(root.left, key);

    return search(root.right, key);
}

public static void main(String[] args) {
    BinarySearchTree tree = new BinarySearchTree();

    /* Let us create following BST
          8
```

```
            /      \
           4       12
          /  \     /  \
         1   7    9   14 */

        tree.insertNode(8);
        tree.insertNode(4);
        tree.insertNode(1);
        tree.insertNode(7);
        tree.insertNode(12);
        tree.insertNode(9);
        tree.insertNode(14);
        System.out.println("Data you are looking for :: " +
                           tree.search(tree.root, 1).key);

        tree.inorder();
        System.out.println("-------------------");

        tree.deleteKey(12);

        tree.inorder();


    }
}
```