**Binary Search Tree**

//In the following program use try catch at appropriate place to handle exceptions

package BinarySearchTree;

import java.util.\*;

public class BST {

public int data;

BST lchild;

BST rchild;

BST() {

}//end of default constructor

BST(int id) {

data = id;

}//end of parameterized constructor

}//end of class

class BinaryST {

BST root;

BinaryST() {

root = null;

}//end of constructor

void insert(int id) {

BST node = new BST(id);

if (root == null) {

System.out.println("Inserting " + id + " as root");

root = node;

}//end of if

else {

BST cur = root;

BST par = null;

while (cur != null) {

par = cur;

if (id < cur.data) {

cur = cur.lchild;

}//end of if

else {

cur = cur.rchild;

}//end of else

}//end of while

if (id < par.data) {

System.out.println("Inserting " + id + " as a left child of " + par.data);

par.lchild = node;

}//end of if

else {

System.out.println("Inserting " + id + " as a right child of " + par.data);

par.rchild = node;

}//end of else

}//end of else

}//end of insert()

public void pre\_Order() {

preOrder(root);

}//end or pre\_Order()

public void preOrder(BST root) {

if (root != null) {

System.out.print(root.data + ",");

preOrder(root.lchild);

preOrder(root.rchild);

}//end of if

}//end of preOrder()

public void in\_Order() {

inOrder(root);

}//end or in\_Order()

public void inOrder(BST root) {

if (root != null) {

inOrder(root.lchild);

System.out.print(root.data + ",");

inOrder(root.rchild);

}//end of if

}//end of inOrder()

public void post\_Order() {

postOrder(root);

}//end or post\_Order()

public void postOrder(BST root) {

if (root != null) {

postOrder(root.lchild);

postOrder(root.rchild);

System.out.print(root.data + ",");

}//end of if

}//end of postOrder()

public void search(int key) {

BST cur = root;

int level = 0;

if (cur == null) {

System.out.println("No tree has been created, so cannot search");

} else {

if (key == root.data) {

System.out.println(key + " fount at level " + level);

} else {

while (true) {

if (key < cur.data) {

level++;

cur = cur.lchild;

if (key == cur.data) {

System.out.println(key + " found at level " + level);

break;

}//end of if

if (cur == null) {

System.out.println(key + " not found");

break;

}//end of if

} else {

level++;

cur = cur.rchild;

if (key == cur.data) {

System.out.println(key + " found at level " + level);

break;

}//end of if

if (cur == null) {

System.out.println(key + " not found");

break;

}//end of if

}//end of else

}//end of while

}//end of else

}//end of else

}//end of search()

void delete(int id) {

//if the tree is not created

if (root == null) {

System.out.println("Tree not created, deletion not possible");

} else {

BST cur = root;

BST par;

int flag = 0;

while (true) {

par = cur;

if (flag == 1) {

cur = cur.lchild;

} else if (flag == 2) {

cur = cur.rchild;

} else {

}//end of else

if (id == cur.data) {

//if the node is leaf

if (cur.lchild == null && cur.rchild == null) {

System.out.println("Element being deleted is::" + cur.data);

if (par.lchild == cur) {

par.lchild = null;

} else {

System.out.println(par.data);

par.rchild = null;

}//end of else

break;

} //if the node has one child (leftchild==null)

else if (cur.lchild == null && cur.rchild != null) {

System.out.println("Element being deleted is::" + cur.data);

if (par.data < cur.data) {

par.rchild = cur.rchild;

break;

} else {

par.lchild = cur.rchild;

break;

}//end of else

} //if the node has one child (rightchild==null)

else if (cur.rchild == null && cur.lchild != null) {

System.out.println("Element being deleted is::" + cur.data);

if (par.data < cur.data) {

par.rchild = cur.lchild;

break;

} else {

par.lchild = cur.lchild;

break;

}//end of else

} //if the node has two children

else {

BST temp = cur;

System.out.println(cur.data + "=" + temp.data);

System.out.println("Element being deleted is::" + cur.data);

temp = temp.lchild;

System.out.println(temp.data);

while (temp.rchild != null) {

par = temp;

temp = temp.rchild;

}//end of while

System.out.println("par=" + par.data);

System.out.println("temp=" + temp.data);

if (temp.lchild == null && temp.rchild == null) {

cur.data = temp.data;

par.rchild = null;

break;

} else {

cur.data = temp.data;

par.rchild = temp.lchild;

break;

}//end of else

}//end of else

} else if (id < cur.data) {

flag = 1;

//cur = cur.lchild;

//par = par.lchild;

} else {

flag = 2;

//cur = cur.rchild;

//par = par.rchild;

}//end of else

}//end of while

}//end of else

}//end of delete()

public static void main(String args[]) {

Scanner inp = new Scanner(System.in);

BinaryST bstobj = new BinaryST();

System.out.println("Inserting Elements::");

bstobj.insert(10);

bstobj.insert(15);

bstobj.insert(5);

bstobj.insert(3);

bstobj.insert(7);

bstobj.insert(12);

bstobj.insert(18);

bstobj.insert(20);

bstobj.insert(6);

System.out.println("Pre\_order Travesal::");

bstobj.pre\_Order();

System.out.println();

System.out.println("In\_order Traversal::");

bstobj.in\_Order();

System.out.println();

System.out.println("Post\_order Traversal::");

bstobj.post\_Order();

System.out.println("\nEnter element to be searched::");

int key = inp.nextInt();

bstobj.search(key);

System.out.println("\nEnter element to be deleted::");

int delkey = inp.nextInt();

bstobj.delete(delkey);

System.out.println("Display in Pre\_Order after deletion::");

bstobj.pre\_Order();

}//end of main

}//end of class