ASSIGNMENT 1

**Name: Soham Makarand Limaye**

**Roll No: 222038**

**PRN: 22111058**

**Batch: B2**

**Aim:** Implement binary search tree and perform following operations: a. Insert, b. Delete, c. Display (Inorder) d. Search e.BFS (Level wise print)

**Objective :**

1. To understand the implementation of the tree using java.
2. to perform the various operation on tree data structure

3)to understand oops in java 4)to understand the binary search tree

**Theory :**

1)In the program first we used the while loop to display the various operation options and after that using the switch case we performed the operations like if the user entered the 1 it would take elements of the tree. For 2 it provides the deletion operation ,3 -- display ,4 --search ,and for 5 it will print the elements level wise. In the switch case it will call a particular function to do that work .

**Code :**

package ADS1;

import java.util.\*; public class ADS1 { class Node{ int

data;

Node leftNode;

Node rightNode;

Node(int data){ this.data=data;

}

}

Node rootNode;

ADS1(){ rootNode=null

;

}

void insertNode(int value) {

Node newNode=new Node(value);

if(rootNode == null) { rootNode=newNode;

}

else {

Node current=rootNode,parent=null; while(true) { parent =current; if(value<current.data)

{ current=current.leftNode; if(current==null)

{ parent.leftNode=newNode; return;

}

}

else { current=current.rightNode; if(current==null)

{ parent.rightNode=newNode; return;

}

}

}

} } boolean searchNode(int value) { Node current=rootNode;

if(rootNode==null) {

System.out.println("\n The BST is empty");

}

else { if(rootNode.data==value)

{

System.out.println("\n Value found");

}

else { while(true) { if(value<current.data)

{ if(current.leftNode==null) { break;

}

else

{ current=current.leftNod e;

}

}

else{ if(current.data==value)

{ break;

}

else if(current.rightNode==null)

{ break;

}

else

{ current=current.rightNod e;

}

}

}

if(current.data==value) {

System.out.println("\n Value found");

}

else {

System.out.println("\n Value not found");

}

}

}

return(current.data==value);

}

Node maxNode(Node node) {

if(node.rightNode!=null) { return

maxNode(node.rightNode);

}

else { return

node;

}

}

Node deleteNode(Node node, int value)

{ if(node==null) { return null;

}

else { if(value<node.data)

{ node.leftNode=deleteNode(node.leftNode,value);

}

else if(value>node.data)

{ node.rightNode=deleteNode(node.rightNode,value);

}

else { if(node.leftNode == null && node.rightNode == null)

{ node = null;

}

else if(node.leftNode == null)

{ node = node.rightNode;

}

else if(node.rightNode == null)

{ node = node.leftNode;

}

else {

Node temp = maxNode(node.leftNode); node.data = temp.data;

node.leftNode = deleteNode(node.leftNode,

temp.data);

}

}

}

return node;

}

void inorderTraversal(Node node) { if(node==null) { System.out.println("The BST is empty"); return;

}

else { if(node.leftNode != null)

{ inorderTraversal(node.leftNode);

}

System.out.print(node.data + "\t"); if(node.rightNode != null)

{ inorderTraversal(node.rightNode);

}

}

}

int height(Node rootNode)

{ if(rootNode==null)

{ return 0;

}

else { int leftHeight =

height(rootNode.leftNode);

int rightHeight = height(rootNode.rightNode); if(leftHeight > rightHeight) { return

(leftHeight +1);

}

else { return

(rightHeight +1);

}

}

}

void printCurrentLevel(Node rootNode, int level)

{ if(rootNode == null) { return;

} if(level == 0) {

System.out.print(rootNode.data + "\t");

} else if(level >

0){ printCurrentLevel(rootNode.leftNode, level - 1); printCurrentLevel(rootNode.rightNode, level - 1);

} } void printLevelOrder()

{ int

h=height(rootNode);

int i; for(i=0;i<=h;i++)

{ printCurrentLevel(rootNode,i);

}

}

public static void main(String[] args) {

ADS1 BST= new ADS1();

Scanner scan= new Scanner(System.in); int choice=0; do { choice=0;

System.out.println("\n\nSelect the Operation that you want

to perform:");

System.out.println("1. Insert node in BST.");

System.out.println("2. Search node.");

System.out.println("3. Delete node from BST BST.");

System.out.println(";4. Display (Inorder traversal in BST.)"); System.out.println(";5. BFS (Level Wise Print.)"); System.out.println("[Enter any other digit to exit.]"); choice = scan.nextInt(); switch(choice) { case 1:

System.out.println("\nEnter data to be inserted:"); int nodeData = scan.nextInt();

BST.insertNode(nodeData);

System.out.println("\nData inserted."); break; case 2:

System.out.println("\nEnter data to be searched:"); int searchData = scan.nextInt();

BST.searchNode(searchData); break; case 3: System.out.println("\nEnter data to be deleted:"); int deleteData = scan.nextInt(); if(BST.searchNode(deleteData) == true) {

System.out.println("\nDeleting data: " + deleteData);

BST.deleteNode(BST.rootNode, deleteData);

}

else {

System.out.println("\nData not found.");

}

break; case 4:

if(BST.rootNode == null) {

System.out.println("\nThe tree is empty.");

}

else {

System.out.println("\nInorder traversal of BST:");

BST.inorderTraversal(BST.rootNode);

}

break; case 5:

BST.printLevelOrder(); default: break;

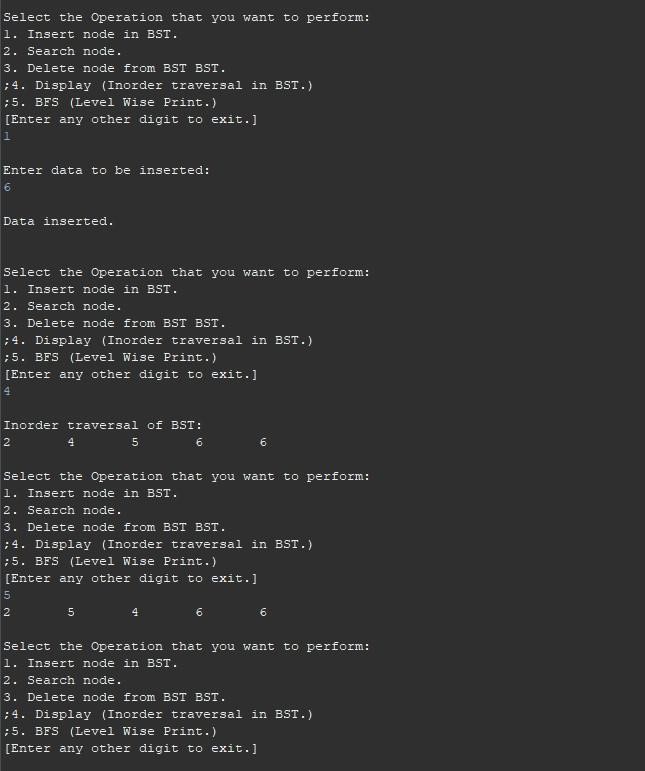
}

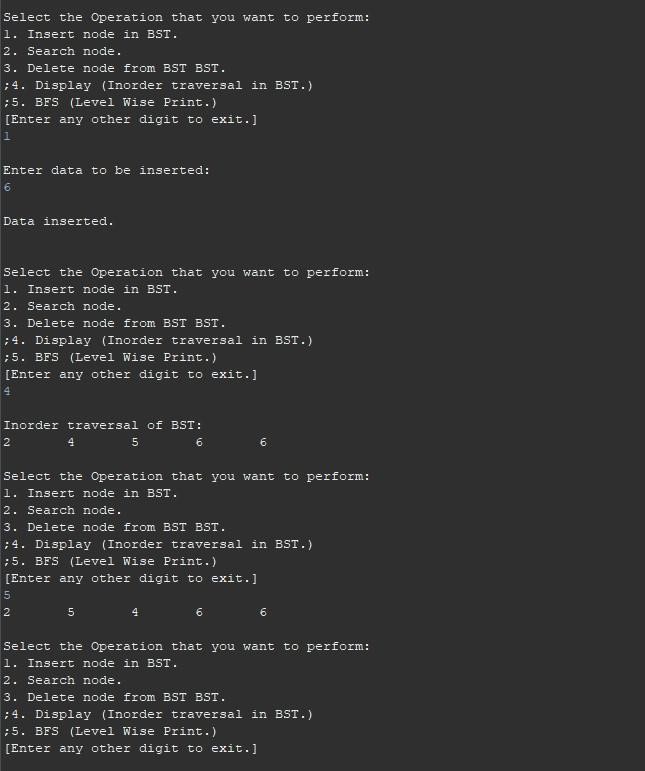
}while(choice<6);

System.out.println("Program finished."); scan.close();

} }

**Output :**





**Conclusion:**

Insert, delete, search, BFS, inorder traversal in BST implemented