For this project assignment, you will solve a problem based on what you have learnt in this course.

**Instructions**

* Write Name and SNU ID of both the group members in the header of this document.
* Assignment submitted after the due date will not be evaluated and a score of zero will be awarded.
* Upload a word version of this document.
* Properly document/comment your code, followed by snapshots of output as desired.

**Due Date and Time:**  **10 pm, November 24, 2019.**

**Submitting this Assignment**

You will submit (upload) this assignment in Blackboard. Email/paper submissions will not be accepted. All students must upload their project individually.

* Name this document as Project\_CSD207-2019\_John\_Bill.doc in case the first names of group members are John and Bill respectively.

**Grading Criteria**

**This assignment has 20 points (with weightage of 10% in your overall 100 points). Points will be awarded as follows:**

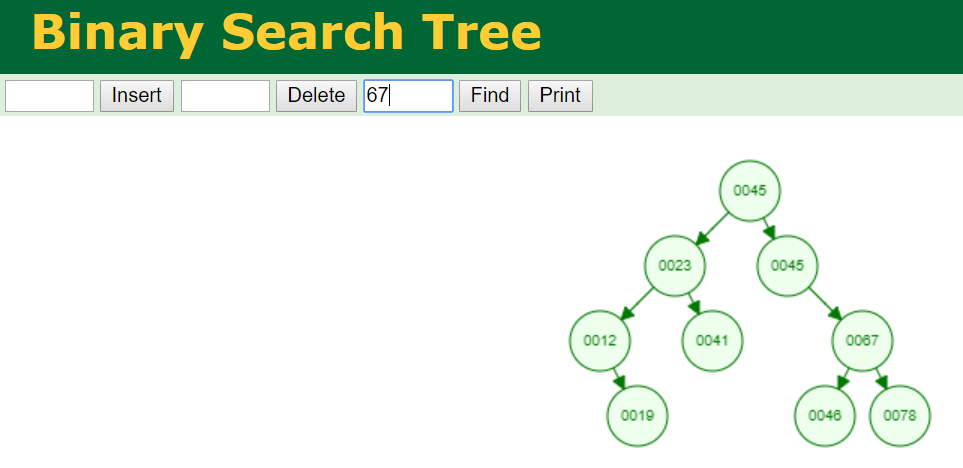
1. Functionality – **14 points**
2. Look and Feel of node creation, deletion and searching implementations – **06 points**

**Project Problem**

Write a java program to create a **binary search tree** (as shown in the figure) that will make use of several Swing components, event handling, graphics and Java Collections Framework to implement. GUI must contain buttons to perform following operations:

1. Insert - to insert a node (element) into the tree
2. Delete - to delete a node from the tree
3. Find- to search an element in the tree
4. Print – to print the sorted list of elements

Program should keep updating the following details at the bottom of the Frame:  
 a. height of the tree  
 b. number of vertices



PROGRM FOR BST :-

package bst;

import java.util.\*; // necessary library imports

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class Project\_bst {

public static void main(String[] args) {

DisplayBinaryTree dbt = new DisplayBinaryTree();

JFrame f = new JFrame("Welcome to your Binary Search Tree!");

f.add(dbt); // adding the binary search tree into the frame

f.setBounds(200, 10, 700, 600); // declaring the dimensions

f.setVisible(true); // making the frame visibility to true

f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

}

class DisplayBinaryTree extends JApplet {

DisplayBinaryTree() {

add(new TreeController(new BinaryTree<Integer>()));

}

}

class TreeController extends JPanel { // class for thr gui of the BST

private BinaryTree<Integer> tree;

// A binary tree to be displayed

private JTextField jtf = new JTextField(5);

private TreeView TreeView = new TreeView();

private JButton Insertion = new JButton("Insert Element");

private JButton deletion = new JButton("Delete Element");

private JButton search = new JButton("Search");

public JLabel label=new JLabel("Number of nodes");

/\*\* Construct a view for a binary tree \*/

TreeController(BinaryTree<Integer> tree) {

this.tree = tree; // Set a binary tree to be displayed

setUI();

}

/\*\* Initialize UI for binary tree \*/

private void setUI() { // creating the GUI for the appliction

this.setLayout(new BorderLayout());

add(TreeView, BorderLayout.CENTER);

JPanel panel = new JPanel(); //creating panel

JPanel p2=new JPanel();

Insertion.setBackground(Color.YELLOW);

deletion.setBackground(Color.YELLOW);

search.setBackground(Color.YELLOW);

panel.add(new JLabel("Enter the element: "));

panel.add(jtf); //adding textfield to the panel

panel.add(Insertion); // adding buttons to the panel

panel.add(deletion);

panel.add(search);

p2.add(label);

p2.setBackground(Color.yellow);

panel.setBackground(Color.cyan); // background colour of the panel

add(panel, BorderLayout.NORTH); // layout of the frame

add(p2, BorderLayout.SOUTH);

// Listener for the Insert button

Insertion.addActionListener(new ActionListener() { //action to be performed if the button "Insertion is pressed

@Override

public void actionPerformed(ActionEvent e) {

int key = Integer.parseInt(jtf.getText());

if (tree.search(key)) { // key is in the tree already

JOptionPane.showMessageDialog(null, key + " cannot be added. It is already in the tree");

}

else {

tree.insert(key); // Insert a new key

TreeView.repaint(); // Redisplay the tree

}

label.setText("Number of nodes: " + tree.size );

}

});

//Listener for the Search Button

search.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int key = Integer.parseInt(jtf.getText());

if (tree.search(key)) { // if the elemene is in the tree already

JOptionPane.showMessageDialog(null, key + " is in the BST");

}

else{

JOptionPane.showMessageDialog(null,

key + " is not in the tree");

}

}

});

// Listener for the Delete button

deletion.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int ele = Integer.parseInt(jtf.getText());

if (!tree.search(ele)) { // element is not in the tree

JOptionPane.showMessageDialog(null, ele + " is not in BST");

}

else {

tree.delete(ele); // Delete a key

TreeView.repaint(); // Redisplay the tree

}

label.setText("Number of nodes: " + tree.size );

}

});

}

class TreeView extends JPanel {

private int radius = 30; // Tree node radius

private int vGap = 70; // Gap between two levels in the tree

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

if (tree.getRoot() != null) {

// Display tree recursively

displayTree(g, tree.getRoot(), getWidth()/2, 30, getWidth()/4); //calling the function for displaying tree

}

}

// Display a subtree rooted at position (x, y)

private void displayTree(Graphics g, BinaryTree.TreeNode root,int x, int y, int hGap) {

// Display the root

g.setColor(Color.BLUE); //setting the color of the oval

g.fillOval(x-radius, y-radius, 2\*radius, 2\*radius); //fillOval creates an oval with filled color

g.setColor(Color.white); //setting the text to be displayed in black

g.drawString(root.element+"", x - 7, y + 4); //seting the location of the text to be displayed inside the oval

if (root.left != null) {

// Draw a line to the left node

g.setColor(Color.RED);

connectLeftChild(g, x - hGap, y + vGap, x, y); //calling functionfor the left half tree

// Drawing the left subtree recursively

displayTree(g, root.left, x - hGap, y + vGap, hGap/2);

}

if (root.right != null) {

// Drawing a line to the right node

g.setColor(Color.RED);

connectRightChild(g, x + hGap, y + vGap, x, y); //right child of a node

// Drawing the right subtree recursively

displayTree(g, root.right, x + hGap, y + vGap, hGap/2);

}

}

/\*\* Connect a parent at (x2, y2) with its left child at (x1, y1) \*/

private void connectLeftChild(Graphics g,int x1, int y1, int x2, int y2) {

double d = Math.sqrt(vGap \* vGap + (x2 - x1) \* (x2 - x1));

int x11 = (int)(x1 + radius \* (x2 - x1) / d); //starting x coordinate

int y11 = (int)(y1 - radius \* vGap / d); //starting y coordinate

int x21 = (int)(x2 - radius \* (x2 - x1) / d); //ending x coordinate for the line

int y21 = (int)(y2 + radius \* vGap / d); //ending y coordinate

g.drawLine(x11, y11, x21, y21);

}

/\*\* Connect a parent at (x2, y2) with

\* its right child at (x1, y1) \*/

private void connectRightChild(Graphics g,int x1, int y1, int x2, int y2) {

double d = Math.sqrt(vGap \* vGap + (x2 - x1) \* (x2 - x1));

int x11 = (int)(x1 - radius \* (x1 - x2) / d);

int y11 = (int)(y1 - radius \* vGap / d);

int x21 = (int)(x2 + radius \* (x1 - x2) / d);

int y21 = (int)(y2 + radius \* vGap / d);

g.drawLine(x11, y11, x21, y21);

}

}

}

class BinaryTree<E extends Comparable<E>>{ // class for the binary tree

protected TreeNode<E> root;

protected int size = 0;

public BinaryTree() {

}

public boolean search(E e) { //function for searching an element in the BST

TreeNode<E> current = root;

while (current != null) {

if (e.compareTo(current.element) < 0) { //applying the binary search algorithm

current = current.left;

}

else if (e.compareTo(current.element) > 0) {

current = current.right;

}

else

return true;

}

return false;

}

public boolean insert(E e) { //function for insertion of a new node

if (root == null) // if the tree is empty

root = createNewNode(e); //node element is created

else {

TreeNode<E> parent = null;

TreeNode<E> current = root;

while (current != null) //recursively adding the new node

if (e.compareTo(current.element) < 0) {

parent = current;

current = current.left;

}

else if (e.compareTo(current.element) > 0) {

parent = current;

current = current.right;

}

else

return false;

if (e.compareTo(parent.element) < 0) // adding the new node

parent.left = createNewNode(e); // the previous node becomes the parent node

else

parent.right = createNewNode(e);

}

size++; //size of the tree

return true;

}

protected TreeNode<E> createNewNode(E e) { //node of the tree

return new TreeNode<E>(e);

}

public static class TreeNode<E extends Comparable<E>> {

E element; // element in the node i.e the number

TreeNode<E> left; // left child

TreeNode<E> right; //right child

public TreeNode(E e) {

element = e;

}

}

public int getSize() {

return size; // size changes after everinng insertion and deletion

}

public TreeNode getRoot() {

return root; // root is the starting node of the tree

}

public boolean delete(E e) { //function for deleting a node

TreeNode<E> parent = null; // recursively finding the location of the node

TreeNode<E> current = root;

while (current != null) {

if (e.compareTo(current.element) < 0) {

parent = current;

current = current.left;

}

else if (e.compareTo(current.element) > 0) {

parent = current;

current = current.right;

}

else

break;

}

if (current == null)

return false;

if (current.left == null) {

if (parent == null) {

root = current.right;

}

else {

if (e.compareTo(parent.element) < 0)

parent.left = current.right;

else

parent.right = current.right;

}

}

else { // if the node to be deleted is not a ending node, but is a parent of another node a new parent node has to be made

TreeNode<E> parentOfRightMost = current;

TreeNode<E> rightMost = current.left;

while (rightMost.right != null) {

parentOfRightMost = rightMost;

rightMost = rightMost.right;

}

current.element = rightMost.element;

if (parentOfRightMost.right == rightMost)

parentOfRightMost.right = rightMost.left;

else

parentOfRightMost.left = rightMost.left;

}

size--; // dereasing the size after deletion

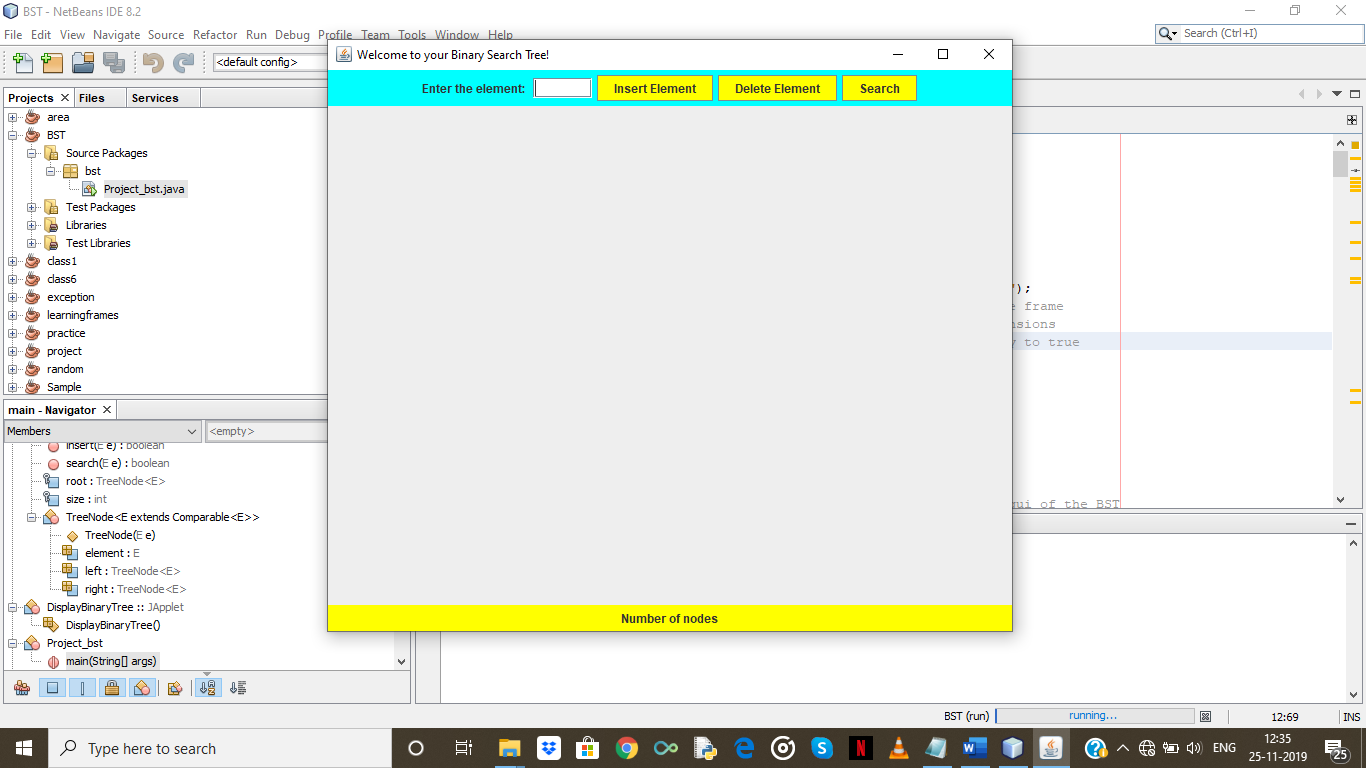
return true;

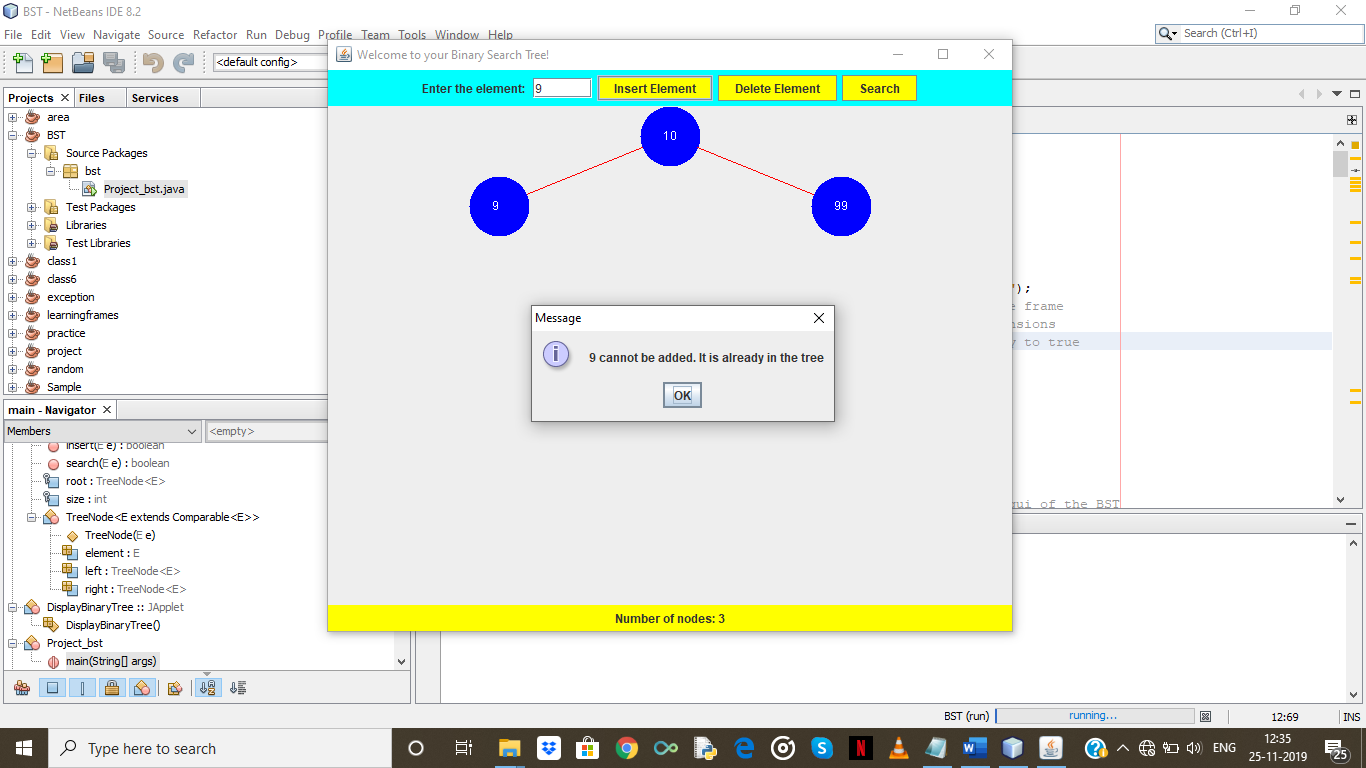
}

}

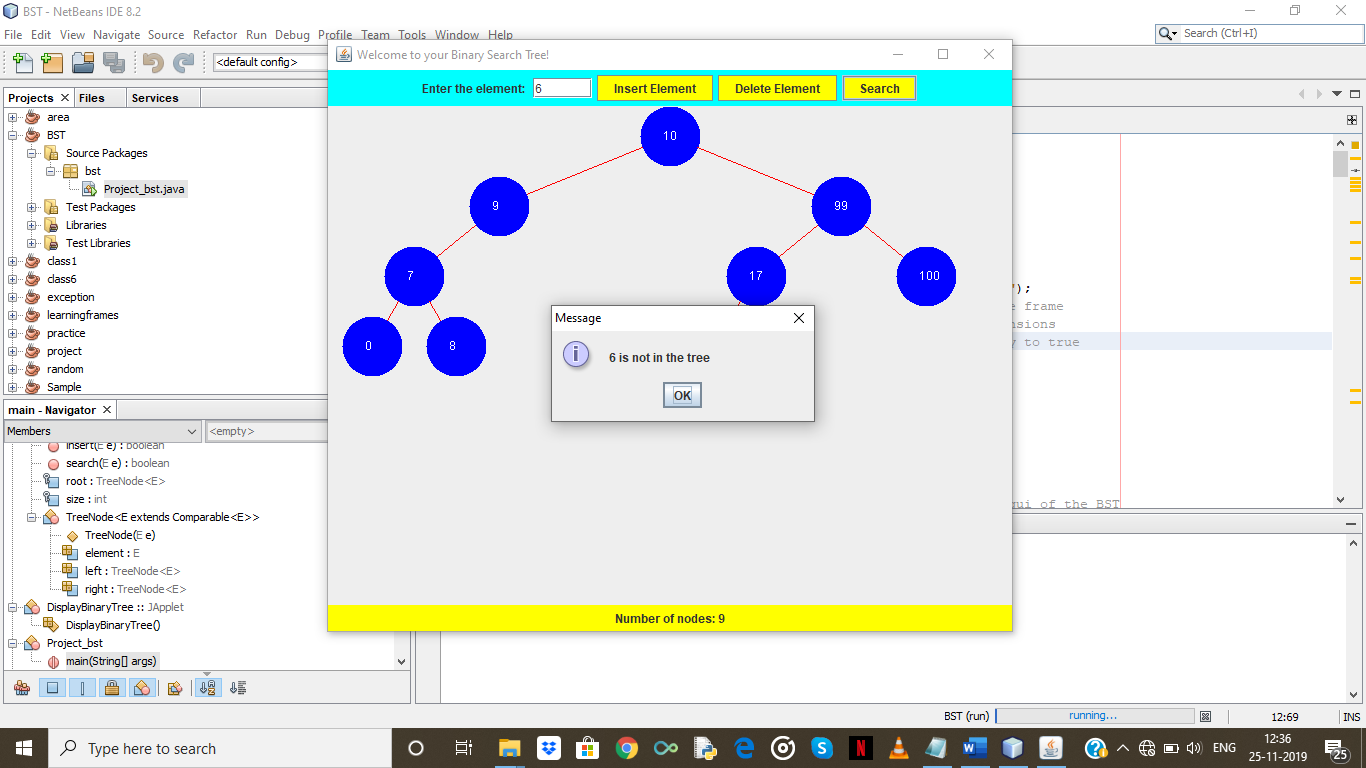
OUTPUTS :-

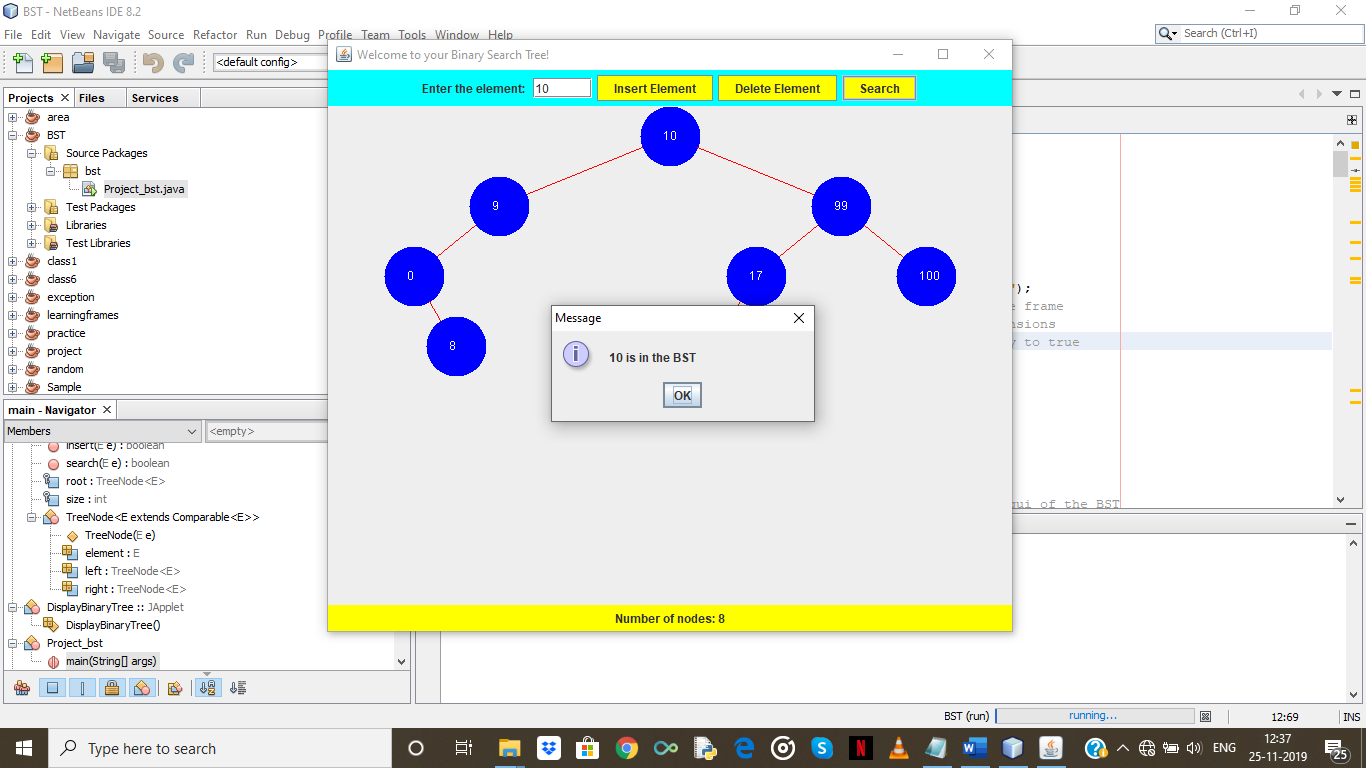
STARTING GUI:



ADDING ELEMENTS:

SEARCHING IN BST :





DELETING NODE 7:

