***BSTTEST.java***

package app;

import java.util.ArrayList;

// package bsttest; // dclj

import java.util.Scanner;

public class BSTTest {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

Integer key;

System.out.println();

System.out.println("Binary Search Tree\n");

Integer[] num ={67, 87, 55, 43, 48, 73, 91, 39, 59, 92, 34, 95};//, 81, 66, 40, 53, 84, 77,23,45,67,56,45,34,21,26,19,17,41, 22,20,24,31,14,11,11,3,18,23,4,5,17,25,

// 67, 87, 55, 43, 48, 73, 91, 39, 59, 92, 34, 95, 81, 66, 40, 53, 84, 77};

for (int i = 0; i < num.length; i++) { System.out.print(num[i]+ " "); }

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

BinarySearchTree<Integer> tree = new BinarySearchTree<>(num);// loads

// >>>>>>>>>> path <<<<<<<<<<

lineSeperator(80, '\*');

System.out.println("\n>>>>>>>>>> PATH: START <<<<<<<<<<\n");

System.out.print("Enter element to search: ");

key = input.nextInt();

// arraylist to handla returned value

ArrayList<Integer> \_retVal = new ArrayList<>(tree.path(key));

if(\_retVal.size() == 0) {

System.out.println("\nSupplied Value Not Found: " + key + "\n");

} // end if

else {

System.out.println("\nSupplied Value Found: " + key + "\n");

if(\_retVal.size() == 1) {

// we are at the root

System.out.println("We are at the root node, it has no ancestor nodes.");

} // end if

else {

System.out.println("Here is the node list from root to supplied value: " + key + "\n");

for(int \_item: \_retVal) { System.out.print(\_item + " "); }

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

} // end else

} // end else

System.out.println(">>>>>>>>>> PATH: END <<<<<<<<<<\n");

// >>>>>>>>>> path <<<<<<<<<<

// >>>>>>>>>> getNumberOfLeaves <<<<<<<<<<

lineSeperator(80, '\*');

System.out.println("\n>>>>>>>>>> getNumberOfLeaves: START <<<<<<<<<<\n");

System.out.println("Number of Leaves found: " + tree.getNumberOfLeaves());

System.out.println("\n>>>>>>>>>> getNumberOfLeaves: END <<<<<<<<<<");

// >>>>>>>>>> getNumberOfLeaves <<<<<<<<<<

// tree.output(); //output tree:inorder, preorder, postorder, # of leaves

//search

System.out.print("Enter element to search:\t");

key = input.nextInt();

System.out.println("tree.search(key) =\t\t" + tree.search(key));

System.out.print("Enter a number for rightsubtree:");

key = input.nextInt();

System.out.println("tree.rightSubTree(key=\t"+tree.rightSubTree(key));

System.out.println();

System.out.println("");

System.out.print("Enter number for leftsubtree:\t");

key = input.nextInt();

System.out.println("tree.leftSubTree(key)=\t"+ tree.leftSubTree(key));

System.out.println();

//delete

System.out.print("\nEnter element to delete:\t");

key = input.nextInt();

tree.delete(key);

System.out.println("The deletion of number\t\t" + key + " is: Go ahead and look, but it's not there :) ");

tree.output();

//complete the code as suggested in Lab document.

//insert

System.out.print("\nEnter an element to insert\t");

key = input.nextInt();

tree.insert(key);

System.out.println("");

System.out.println("The insertion of number " + key + " is:");

tree.output();

System.out.print("Enter number for path:\t\t");

key = input.nextInt();

System.out.println(tree.path(key));

/\*

System.out.print("Enter a number for inorderPredecessor:\t");

key = input.nextInt();

System.out.println("number of inorderPredecessor:" + tree.inorderPredecessor(key));

\*/

System.out.print("Enter a number for inorderPredecessor:\t");

key = input.nextInt();

System.out.println("number of inorderPredecessor:" + tree.inOrder2(key));

input.close(); // dcljr

}//main

/\*\*

\*

\* <p><strong><em>Description: </em></strong>adds a line of characters for console display</p>

\*

\* <p><strong><em>Method Name: </em></strong>lineSeperator</p>

\*

\* <p><strong><em>Method Notes: </em></strong>none</p>

\*

\* <p><strong><em>Pre-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Post-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>

\* <p><strong><em>Start Date: </em></strong>04.21.2020</p>

\*

\* @param N how many characters to add

\* @param ch character in line

\*/

public static void lineSeperator(int N, char ch) {

if(N > 1) {

System.out.print(ch);

lineSeperator(N - 1, ch);

} // end if

else { System.out.println(""); } // end else

} // end lineSeperator

}//class

***BinarySearchTree.java***

package app; // dcljr

// import java.io.\*; // dcljr

import java.util.\*;

public class BinarySearchTree<E extends Comparable<E>> extends AbstractTree<E> {

protected TreeNode<E> root;

protected int size = 0;

// >>>>>>>>>>>>>>>>>>> PROJECT CODE <<<<<<<<<<<<<<<<<<<

/\*\*

\*

\* <p><strong><em>Description: </em></strong>none</p>

\*

\* <p><strong><em>Method Name: </em></strong>path</p>

\*

\* <p><strong><em>Method Notes: </em></strong>Returns an ArrayList containing elements in the path from the root leading to the specified element, returns an empty ArrayList if no such element exists.</p>

\*

\* <p><strong><em>Pre-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Post-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>

\* <p><strong><em>Start Date: </em></strong>04.21.2020</p>

\*

\* @param e element to look for

\* @return ArrayList with found data, empty ArrayList if nothing found

\*/

public ArrayList<E> path(E e){

// variables

java.util.ArrayList<E> list = new java.util.ArrayList<>();

TreeNode<E> current = root; // Start from the root

//implement the code here as in search method.

// loop the tree

while (current != e) {

// if we hit null e is not in the tree so reset list and get out

if(current == null) {

list.clear(); //clear the list

break; // get out

} // end if

else {

// echo to see the nodes that are searched

// System.out.println(current.element.toString());

// build our array of searched nodes

list.add(current.element);

if (e.compareTo(current.element) < 0) { current = current.left; } // go left

else if (e.compareTo(current.element) > 0) { current = current.right; } // go right

else { break; } // found it so get out

} // end else

}

// echo to test list size...any value other than 0 and we found e

// System.out.println("List size: " + list.size());

return list; // Return an array of elements

} // end path

/\*\*

\*

\* <p><strong><em>Description: </em></strong>none</p>

\*

\* <p><strong><em>Method Name: </em></strong>getNumberOfLeaves</p>

\*

\* <p><strong><em>Method Notes: </em></strong>Returns the number of leaf nodes in this tree, returns 0 if tree is empty</p>

\*

\* <p><strong><em>Pre-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Post-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>

\* <p><strong><em>Start Date: </em></strong>04.21.2020</p>

\*

\* @return number of leaf nodes or 0 if empty

\*/

public int getNumberOfLeaves(){

// variables

TreeNode<E> current = root; // Start from the root

return getNumberOfLeaves(current);

} // end getNumberOfLeaves

/\*\*

\*

\* <p><strong><em>Description: </em></strong>helper method for same named method</p>

\*

\* <p><strong><em>Method Name: </em></strong>getNumberOfLeaves</p>

\*

\* <p><strong><em>Method Notes: </em></strong>helper method to recursivelly count tree leafs</p>

\*

\* <p><strong><em>Pre-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Post-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>

\* <p><strong><em>Start Date: </em></strong>04.21.2020</p>

\*

\* @param root root of tree to search

\* @return recursive count of leaf numbers

\*/

public int getNumberOfLeaves(TreeNode<E> root) {

if(root == null) { return 0; } // base case

if(root.left == null && root.right == null) { return 1; } // leaf

else {

return getNumberOfLeaves(root.left)

+ getNumberOfLeaves(root.right);

} // end else

} // end helper getNumberOfLeaves

/\*\*

\*

\* <p><strong><em>Description: </em></strong>EXTRA CREDIT</p>

\*

\* <p><strong><em>Method Name: </em></strong>inorderPredecessor</p>

\*

\* <p><strong><em>Method Notes: </em></strong>Returns the inorder predecessor of the specified element, returns null if tree is empty or element 'e' is not in the tree.</p>

\*

\* <p><strong><em>Pre-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Post-Conditions: </em></strong>none</p>

\*

\* <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>

\* <p><strong><em>Start Date: </em></strong>04.21.2020</p>

\*

\* @param e element to look for

\* @return null if tree is empty or "e" is not in the tree, inorder predecessor otherwise

\*/

public E inorderPredecessor(E e){

// if we get here assumpiton is nothing was found or invalid search

return null;

} // end inorderPredecessor

// >>>>>>>>>>>>>>>>>>> PROJECT CODE <<<<<<<<<<<<<<<<<<<

/\*\* Create a default binary tree \*/

public BinarySearchTree() { }

/\*\* Create a binary tree from an array of objects \*/

public BinarySearchTree(E[] objects) {

for (int i = 0; i < objects.length; i++)

insert(objects[i]);

}

/\*\* Returns true if the element is in the tree \*/

public boolean search(E e) {

TreeNode<E> current = root; // Start from the root

while (current != null) {

if (e.compareTo(current.element) < 0) { current = current.left; }

else if (e.compareTo(current.element) > 0) { current = current.right; }

else // element matches current.element

return true; // Element is found

}

return false;

}

/\*\* Insert element o into the binary tree

\* Return true if the element is inserted successfully.

\* Uses an iterative algorithm

\*/

public boolean insert(E e) {

if (root == null)

root = createNewNode(e); // Create a new root

else {

// Locate the parent node

TreeNode<E> parent = null;

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

return false; // Duplicate node not inserted

// Create the new node and attach it to the parent node

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

parent.left = createNewNode(e);

else

parent.right = createNewNode(e);

}

size++;

return true; // Element inserted

}

protected TreeNode<E> createNewNode(E e) { return new TreeNode<E>(e); }

/\*\* Inorder traversal from the root\*/

public void inorder() { inorder(root); }

/\*\* Inorder traversal from a subtree \*/

protected void inorder(TreeNode<E> root) {

if (root == null) return;

inorder(root.left);

System.out.print(root.element + " ");

inorder(root.right);

}

/\*\* Postorder traversal from the root \*/

public void postorder() { postorder(root); }

/\*\* Postorder traversal from a subtree \*/

protected void postorder(TreeNode<E> root) {

if (root == null) return;

postorder(root.left);

postorder(root.right);

System.out.print(root.element + " ");

}

/\*\* Preorder traversal from the root \*/

public void preorder() { preorder(root); }

/\*\* Preorder traversal from a subtree \*/

protected void preorder(TreeNode<E> root) {

if (root == null) return;

System.out.print(root.element + " ");

preorder(root.left);

preorder(root.right);

}

/\*\* Inner class tree node \*/

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

E element;

TreeNode<E> left;

TreeNode<E> right;

public TreeNode(E e) { element = e; }

}

/\*\* Get the number of nodes in the tree \*/

public int getSize() { return size; }

/\*\* Returns the root of the tree \*/

// public TreeNode getRoot() { return root; } // dcljr

public TreeNode<E> getRoot() { return root; }

// dcljr

// commented out so code can be added at the top of the file. this is stubbed here but we are requred to modify

// dcljr

/\*\* Returns an ArrayList containing elements in the path from the root leading to the specified element, returns an empty ArrayList if no such element exists. \*/

// public ArrayList<E> path(E e){

// java.util.ArrayList<E> list = new java.util.ArrayList<>();

// TreeNode<E> current = root; // Start from the root

// //implement the code here as in search method.

// return list; // Return an array of elements

// }

// dcljr

// commented out, project requirement added to top of file.

// /\* Returns the number of leaf nodes in this tree, returns 0 if tree is empty\*/

// public int getNumberOfLeaves(){

// //left for you to implement in Lab 7

// return 0; // dcljr

// }

/\* Returns an ArrayList containing all elements in preorder of the specified element’s left sub-tree, returns an empty ArrayList if no such element exists. \*/

public ArrayList<E> leftSubTree(E e){

return null; // dcljr

//left for you to implement in Lab 7

}

/\* Returns an ArrayList containing all elements in preorder of the specified element’s right sub-tree, returns an empty ArrayList if no such element exists. \*/

public ArrayList<E> rightSubTree(E e){

return null; // dcljr

//left for you to implement in Lab 7

}

//dcljr

// commented out. added to top of file for project requirement

/\* Returns the inorder predecessor of the specified element, returns null if tree is empty or element 'e' is not in the tree. \*/

// public E inorderPredecessor(E e){

// return e; // dcljr

// //left for you to implement in Lab 7

// }

/\*\* Delete an element from the binary tree.

\* Return true if the element is deleted successfully

\* Return false if the element is not in the tree \*/

public boolean delete(E e) {

// Locate the node to be deleted and also locate its parent node

TreeNode<E> parent = null;

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; // Element is in the tree pointed by current

}

if (current == null)

return false; // Element is not in the tree

// Case 1: current has no left children

if (current.left == null) {

// Connect the parent with the right child of the current node

if (parent == null) {

root = current.right;

}

else {

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

parent.left = current.right;

else

parent.right = current.right;

}

}

else {

// Case 2 & 3: The current node has a left child

// Locate the rightmost node in the left subtree of

// the current node and also its parent

TreeNode<E> parentOfRightMost = current;

TreeNode<E> rightMost = current.left;

while (rightMost.right != null) {

parentOfRightMost = rightMost;

rightMost = rightMost.right; // Keep going to the right

}

// Replace the element in current by the element in rightMost

current.element = rightMost.element;

// Eliminate rightmost node

if (parentOfRightMost.right == rightMost)

parentOfRightMost.right = rightMost.left;

else

// Special case: parentOfRightMost == current

parentOfRightMost.left = rightMost.left;

}

size--;

return true; // Element inserted

}

/\*\* Obtain an iterator. Use inorder. \*/

// public java.util.Iterator iterator() { return inorderIterator(); } // dcljr

public java.util.Iterator<E> iterator() { return inorderIterator(); }

/\*\* Obtain an inorder iterator \*/

// public java.util.Iterator inorderIterator() { return new InorderIterator(); } // dcl jr

public java.util.Iterator<E> inorderIterator() { return new InorderIterator(); }

// Inner class InorderIterator

// class InorderIterator implements java.util.Iterator { // dcljr

class InorderIterator implements java.util.Iterator<E> {

// Store the elements in a list

private java.util.ArrayList<E> list = new java.util.ArrayList<E>();

private int current = 0; // Point to the current element in list

// Traverse binary tree and store elements in list

public InorderIterator() { inorder(); }

/\*\* Inorder traversal from the root\*/

private void inorder() { inorder(root); }

/\*\* Inorder traversal from a subtree \*/

private void inorder(TreeNode<E> root) {

if (root == null)return;

inorder(root.left);

list.add(root.element);

inorder(root.right);

}

/\*\* Next element for traversing? \*/

public boolean hasNext() {

if (current < list.size())

return true;

return false;

}

/\*\* Get the current element and move cursor to the next \*/

// public Object next() { return list.get(current++); } // dcljr

public E next() { return list.get(current++); }

/\*\* Remove the current element and refresh the list \*/

public void remove() {

delete(list.get(current)); // Delete the current element

list.clear(); // Clear the list

inorder(); // Rebuild the list

}

}

/\*\* Remove all elements from the tree \*/

public void clear() {

root = null;

size = 0;

}

// dcljr

public String inOrder2(Integer key) { return null; }

public void output() { } // dcljr

}

AbstractTree.java

package app;

// package bsttest; // dcljr

public abstract class AbstractTree<E extends Comparable<E>>implements Tree<E> {

/\*\* Inorder traversal from the root\*/

public void inorder() { }

/\*\* Postorder traversal from the root \*/

public void postorder() { }

/\*\* Preorder traversal from the root \*/

public void preorder() { }

/\*\* Return true if the tree is empty \*/

public boolean isEmpty() { return getSize() == 0; }

/\*\* Return an iterator to traverse elements in the tree \*/

// public java.util.Iterator iterator() { return null; } // dcljr

public java.util.Iterator<E> iterator() { return null; }

}

***Tree.java***

package app;

// package bsttest; // dcljr

public interface Tree<E extends Comparable<E>> {

/\*\* Return true if the element is in the tree \*/

public boolean search(E e);

/\*\* Insert element o into the binary tree

\* Return true if the element is inserted successfully \*/

public boolean insert(E e);

/\*\* Delete the specified element from the tree

\* Return true if the element is deleted successfully \*/

public boolean delete(E e);

/\*\* Inorder traversal from the root\*/

public void inorder();

/\*\* Postorder traversal from the root \*/

public void postorder();

/\*\* Preorder traversal from the root \*/

public void preorder();

/\*\* Get the number of nodes in the tree \*/

public int getSize();

/\*\* Return true if the tree is empty \*/

public boolean isEmpty();

/\*\* Return an iterator to traverse elements in the tree \*/

// public java.util.Iterator iterator(); // dcljr

public java.util.Iterator<E> iterator();

}

***Console Output***

Binary Search Tree

67 87 55 43 48 73 91 39 59 92 34 95

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

>>>>>>>>>> PATH: START <<<<<<<<<<

Enter element to search: 95

Supplied Value Found: 95

Here is the node list from root to supplied value: 95

67 87 91 92 95

>>>>>>>>>> PATH: END <<<<<<<<<<

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

>>>>>>>>>> getNumberOfLeaves: START <<<<<<<<<<

Number of Leaves found: 5

>>>>>>>>>> getNumberOfLeaves: END <<<<<<<<<<

Enter element to search: 5

tree.search(key) = false

Enter a number for rightsubtree:5

tree.rightSubTree(key= null

Enter number for leftsubtree: 5

tree.leftSubTree(key)= null

Enter element to delete: 5

The deletion of number 5 is: Go ahead and look, but it's not there :)

Enter an element to insert 5

The insertion of number 5 is:

Enter number for path: 5

[67, 55, 43, 39, 34, 5]

Enter a number for inorderPredecessor: 5

number of inorderPredecessor:null