Assignment 2
COSC2007: Data Structure II
Jeel Tikiwala
239659420

Exercise No:
1 - Construct a binary search tree to keep the records of the student's names. You can insert the name of students that should follow the rules of binary search tree until press character 'N'.

Algorithm/ Pseudocode:

```
// Define a class for the nodes of the BST
// Constructor to initialize the node with a name
// Define a class for the binary search tree
// A method to insert a new node with a given name into the tree
// If tree is empty, make the new node the root
// Otherwise, start from the root and traverse the tree
// Compare the name to the name of the active node.
// If the name is smaller than the current node's name, go to the left subtree
// If the name is larger than the current node's name, go to the right subtree
```

Code:

INSERT METHOD:

```
    //insert method
    public static Node insert(Node root, String key)
{
        if(root == null) {
            root = new Node(key);
            return root;
        }

        //if key is
        if(key.compareTo(root.data) < 0) {
            root.left = insert(root.left, key);
        }

        else {
            root.right = insert(root.right, key);
        }

        return root;
    }
```

MAIN METHOD:

```
// Q1: Construct a binary search tree
        String studentNames[] = {"Jeel", "Rishit",
"Bhakti", "Shubh", "Ashwin", "Shaan", "Diya"};
        Node root = null;

        //Multiple insert through for loop
        for(int i=0; i<studentNames.length; i++) {
            root = insert(root, studentNames[i]);
        }
```

2 - Perform the search operation to find the particular student.

Algorithm/ Pseudocode:

// A method to search for a node with a given name in the tree
// Navigate the tree starting at the root.
// Compare the name to the name of the active node.
// If the name is smaller than the current node's name, go to the left subtree
// If the name is larger than the current node's name, go to the right subtree
// If the name is equal to the current node's name, return true

Code:

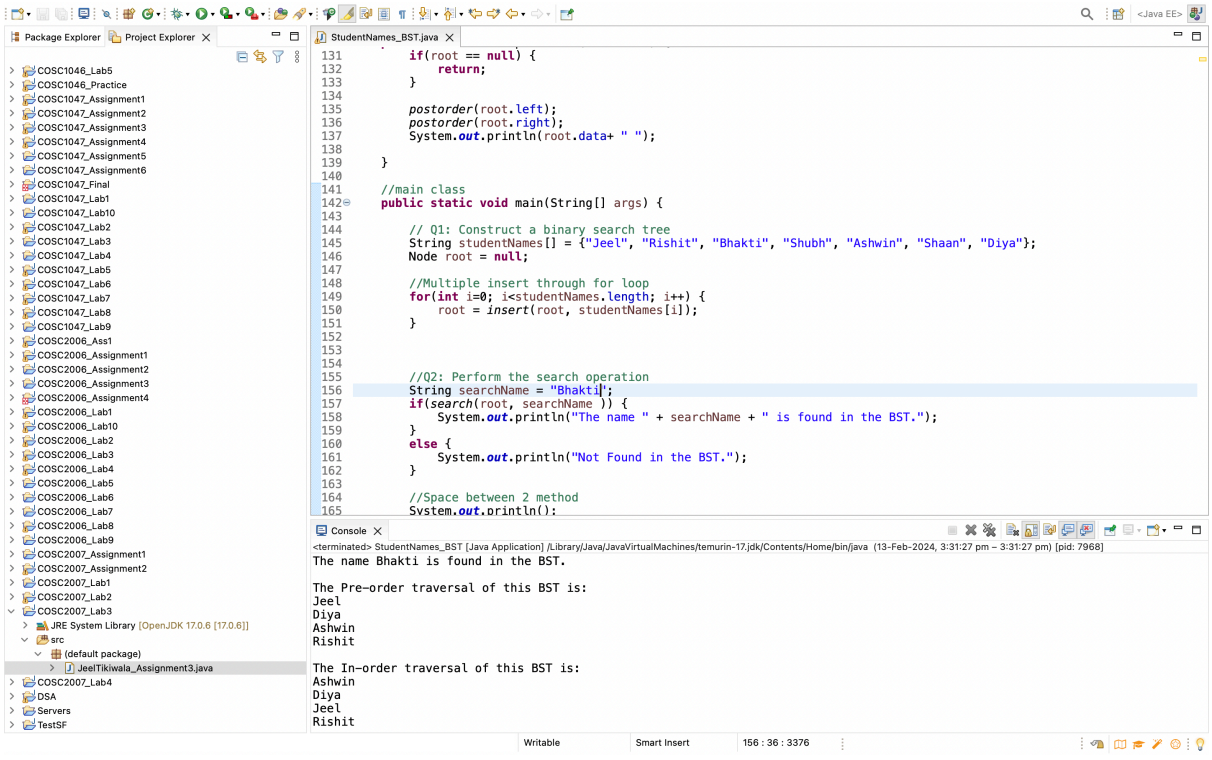
SEARCH METHOD:

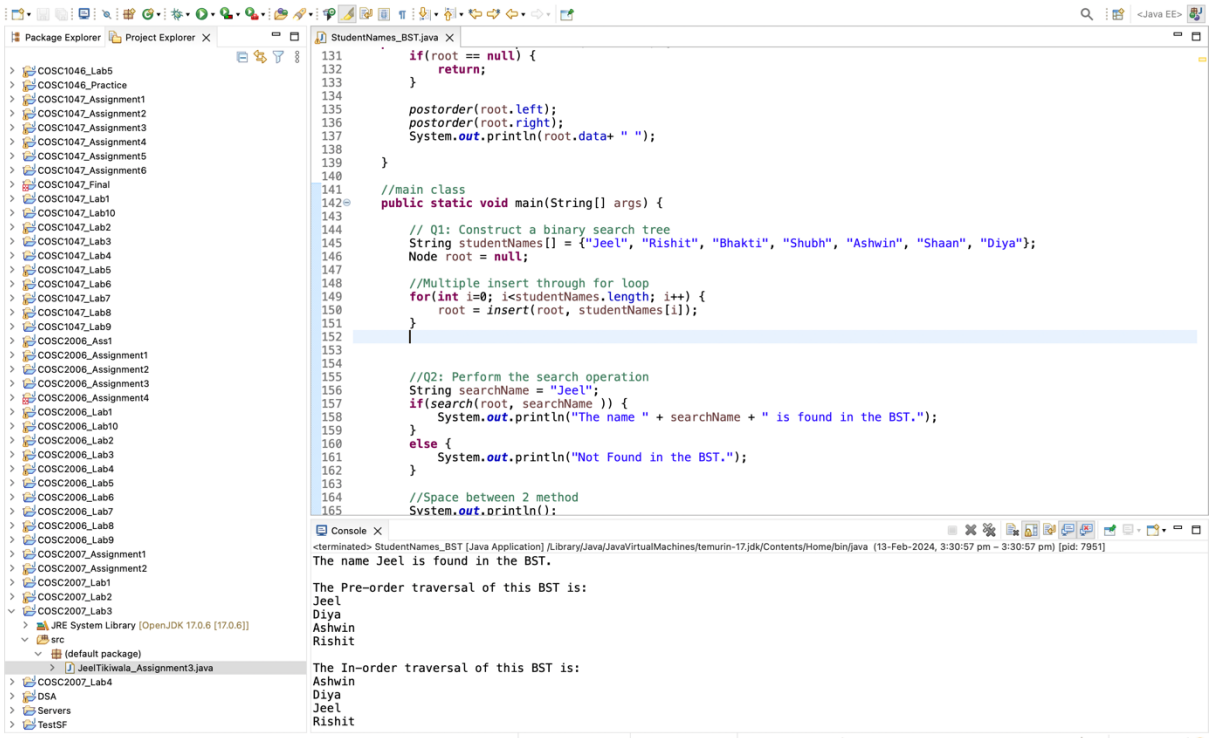
```
    //search method
    public static boolean search(Node root, String
studentNames) {
        if (root == null) {
            return false;
        }

        int comparison =
studentNames.compareTo(root.data);

        if (comparison < 0) {
            // Search in the left subtree
            return search(root.left, studentNames);
        } else if (comparison > 0) {
            // Search in the right subtree
            return search(root.right, studentNames);
        } else {
            // If root is equal to studentNames
            return true;
        }
    }
```

MAIN METHOD:

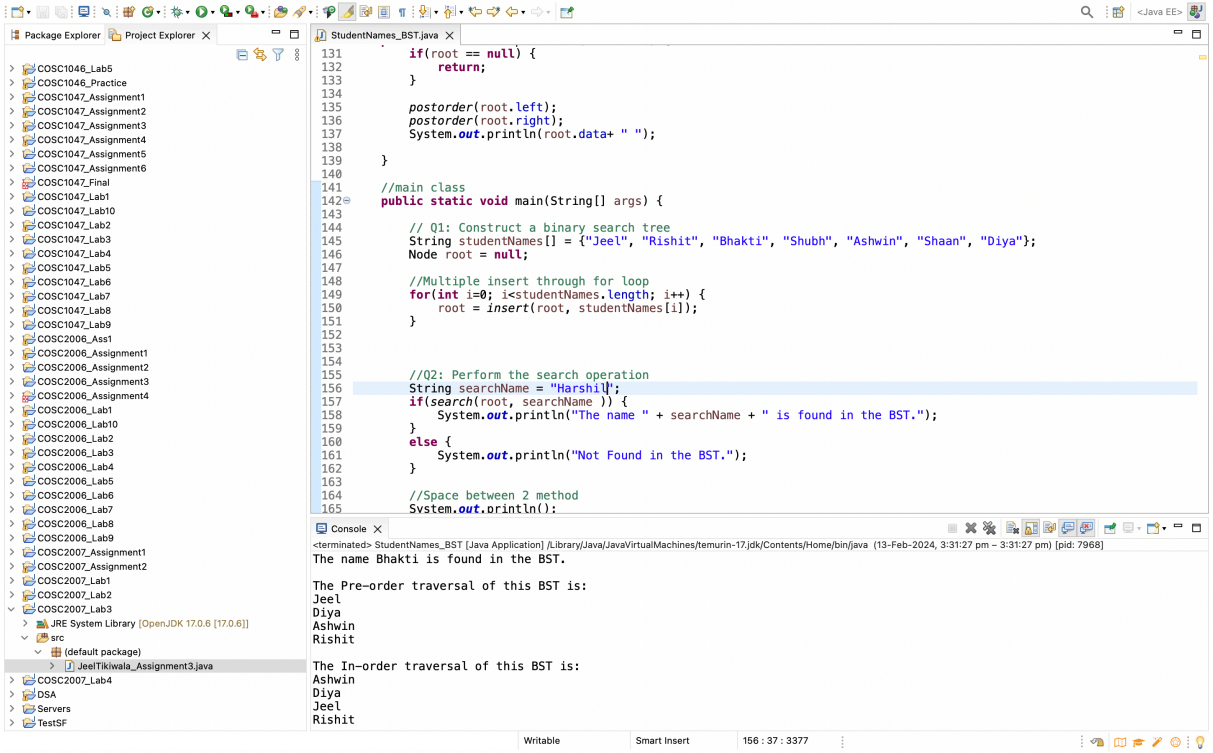
```
        //Q2: Perform the search operation
        String searchName = "Jeel";
        if(search(root, searchName )) {
            System.out.println("The name " +
searchName + " is found in the BST.");
        }
        else {
            System.out.println("Not Found in the
BST.");
        }
```

OUTPUT:





The student “Harshil” is not in the BST” so it returned not found.



3- Delete any one leaf node student

Algorithm/ Pseudocode

// A method to delete a node with a given name from the tree
// Navigate the tree starting at the root.
// If the name is smaller than the current node's name, go to the left subtree
// If the name is larger than the current node's name, go to the right subtree
// Case 1: delete the node that has no children
// If left and right nodes are null, it's a leaf

Code

DELETE METHOD:

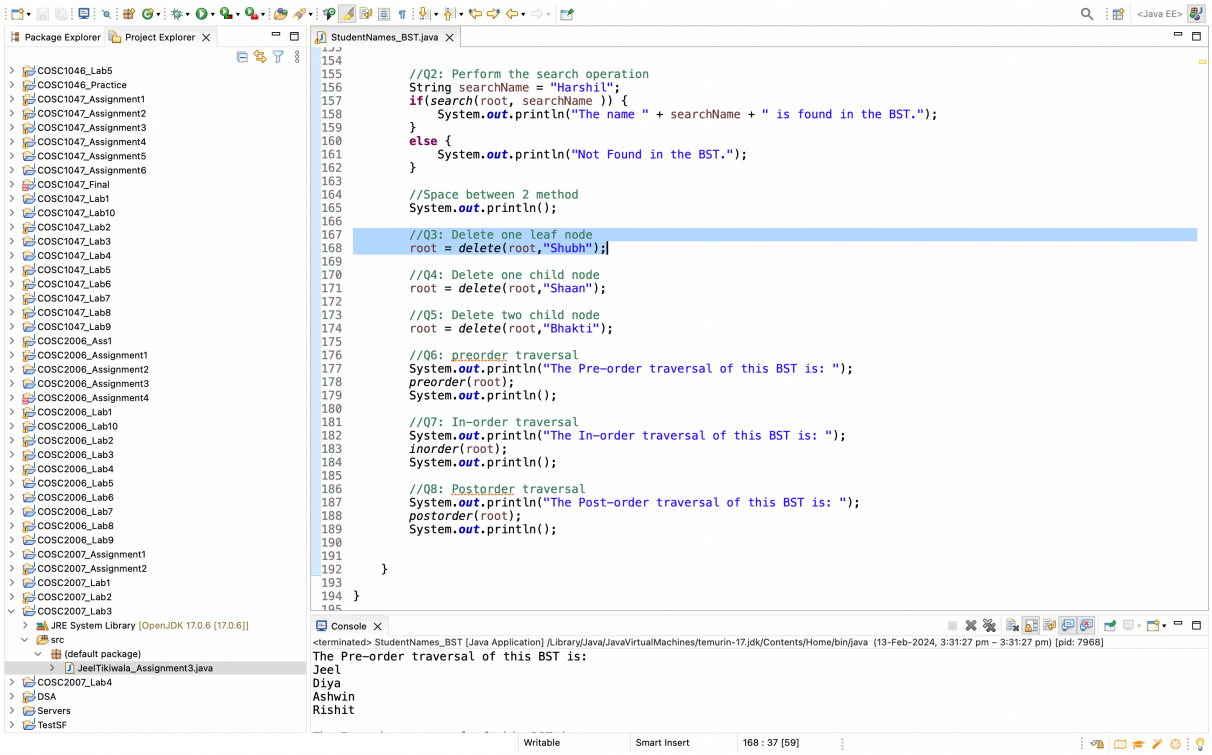
```
//delete a node
    public static Node delete(Node root, String
studentNames) {
        int comparison =
studentNames.compareTo(root.data);

        if(comparison < 0) {
            root.left =
delete(root.left,studentNames);
        } else if (comparison > 0) {
            // Search in the right subtree
            root.right = delete(root.right,
studentNames);
        } else {
            // we searched and reached at the node
        //Case 1: 1st NO child – leaf node :: delete
node and replace with null
        if(root.left == null && root.right == null) {
            return null;
        }

        //Case 2: 2nd One child :: delete node and
replace with child node
        if(root.left == null) {
            return root.right;
        }
        else if(root.right == null) {
            return root.left;
        }
```

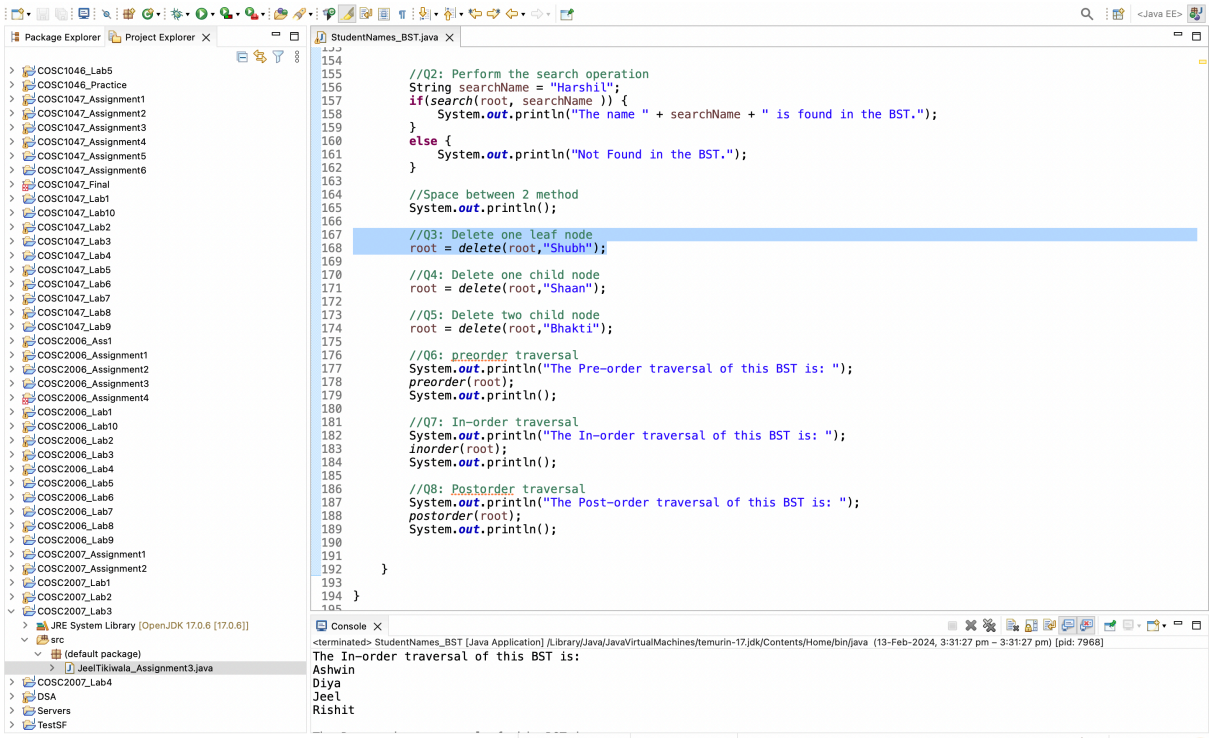
```
        //Case 3: 3rd Two child:: replace value with inorder succesor, delete the node for inorder succesor
        Node IS = inorderSuccesssor(root.right);
        root.data = IS.data;
        root.right = delete(root.right, IS.data);
        }

        return root;
    }
    //inorder succesor fucntion
    public static Node inorderSuccesssor(Node root) {
        while(root.left != null) {
            root = root.left;
        }

        return root;
    }
```

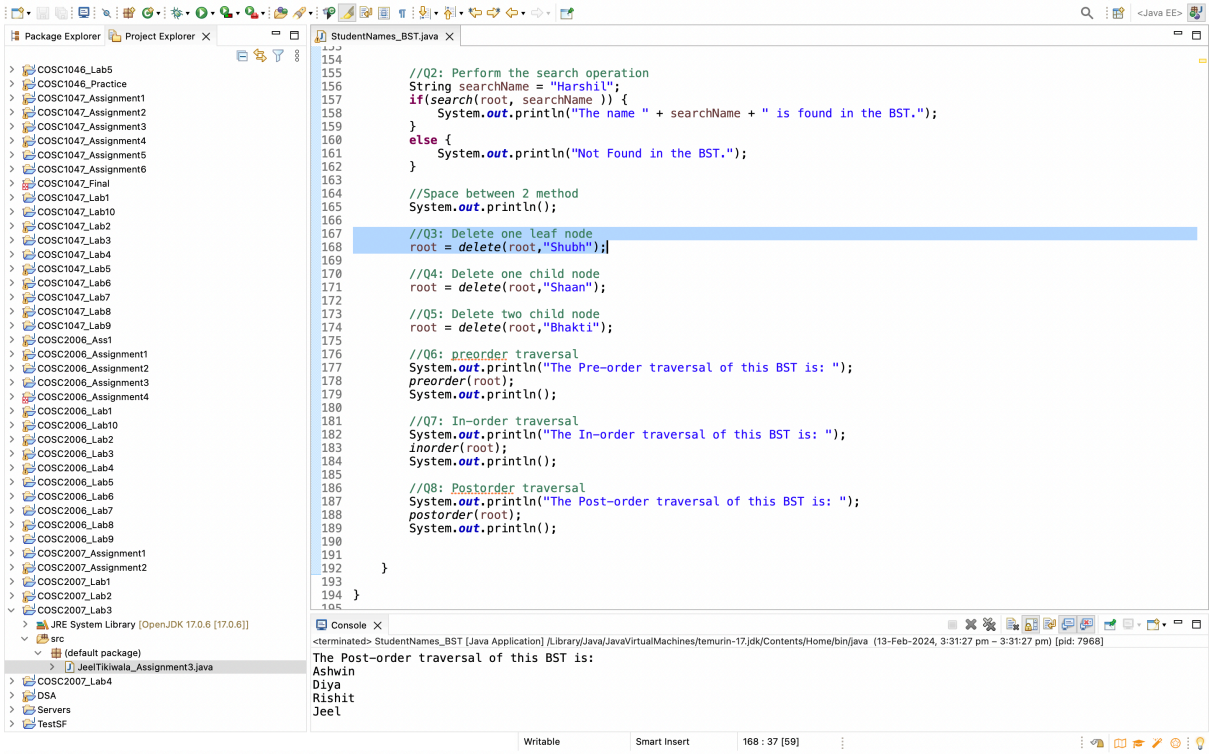
MAIN METHOD:

```
        //Q3: Delete one leaf node
        root = delete(root,"Shubh");
```

OUTPUT:







Conclusion:

4- Delete any one student with one child node

Algorithm/ Pseudocode

// A method to delete a node with a given name from the tree

// Navigate the tree starting at the root.

// If the name is smaller than the current node's name, go to the left subtree
// If the name is larger than the current node's name, go to the right subtree
// If left is null and right is null, delete the node and replace with child node

Code

DELETE METHOD:

```
    //delete a node
    public static Node delete(Node root, String
studentNames) {
        int comparison =
studentNames.compareTo(root.data);

        if(comparison < 0) {
            root.left =
delete(root.left,studentNames);
        } else if (comparison > 0) {
            // Search in the right subtree
            root.right = delete(root.right,
studentNames);
        } else {
            // we searched and reached at the node
        //Case 1: 1st NO child – leaf node :: delete
node and replace with null
        if(root.left == null && root.right == null) {
            return null;
        }

        //Case 2: 2nd One child :: delete node and
replace with child node
        if(root.left == null) {
            return root.right;
        }
        else if(root.right == null) {
            return root.left;
        }

        //Case 3: 3rd Two child:: replace value with
inorder succesor, delete the node for inorder
succesor
        Node IS = inorderSuccesssor(root.right);
        root.data = IS.data;
        root.right = delete(root.right, IS.data);
```

```
        }

        return root;
    }

    //inorder succesor fucntion
    public static Node inorderSuccesssor(Node root) {
        while(root.left != null) {
            root = root.left;
        }

        return root;
    }
```

MAIN METHOD:

```
        //Q4: Delete one child node
        root = delete(root,"Shaan");
```

5 - Delete any one student with two child nodes.

Algorithm/ Pseudocode

// A method to delete a node with a given name from the tree
// Navigate the tree starting at the root.
// If the name is smaller than the current node's name, go to the left subtree
// If the name is larger than the current node's name, go to the right subtree
// replace value with inorder successor, delete the node for inorder successor

Code

Delete method

```
    //delete a node
    public static Node delete(Node root, String
studentNames) {
        int comparison =
studentNames.compareTo(root.data);

        if(comparison < 0) {
            root.left =
delete(root.left,studentNames);
        } else if (comparison > 0) {
            // Search in the right subtree
```

```
            root.right = delete(root.right,
studentNames);
        } else {
            // we searched and reached at the node
        //Case 1: 1st NO child - leaf node :: delete
node and replace with null
        if(root.left == null && root.right == null) {
            return null;
        }

        //Case 2: 2nd One child :: delete node and
replace with child node
        if(root.left == null) {
            return root.right;
        }
        else if(root.right == null) {
            return root.left;
        }

        //Case 3: 3rd Two child:: replace value with
inorder succesor, delete the node for inorder
succesor
        Node IS = inorderSuccesssor(root.right);
        root.data = IS.data;
        root.right = delete(root.right, IS.data);
        }

        return root;
    }

    //inorder succesor fucntion
    public static Node inorderSuccesssor(Node root) {
        while(root.left != null) {
            root = root.left;
        }

        return root;
    }
```

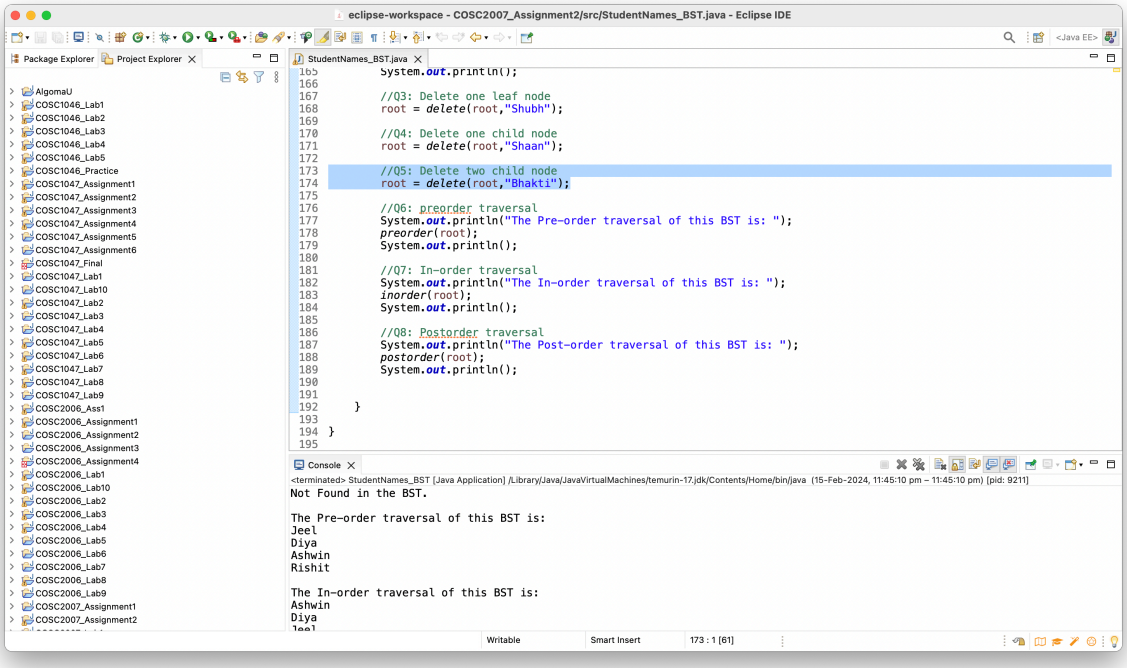
MAIN METHOD

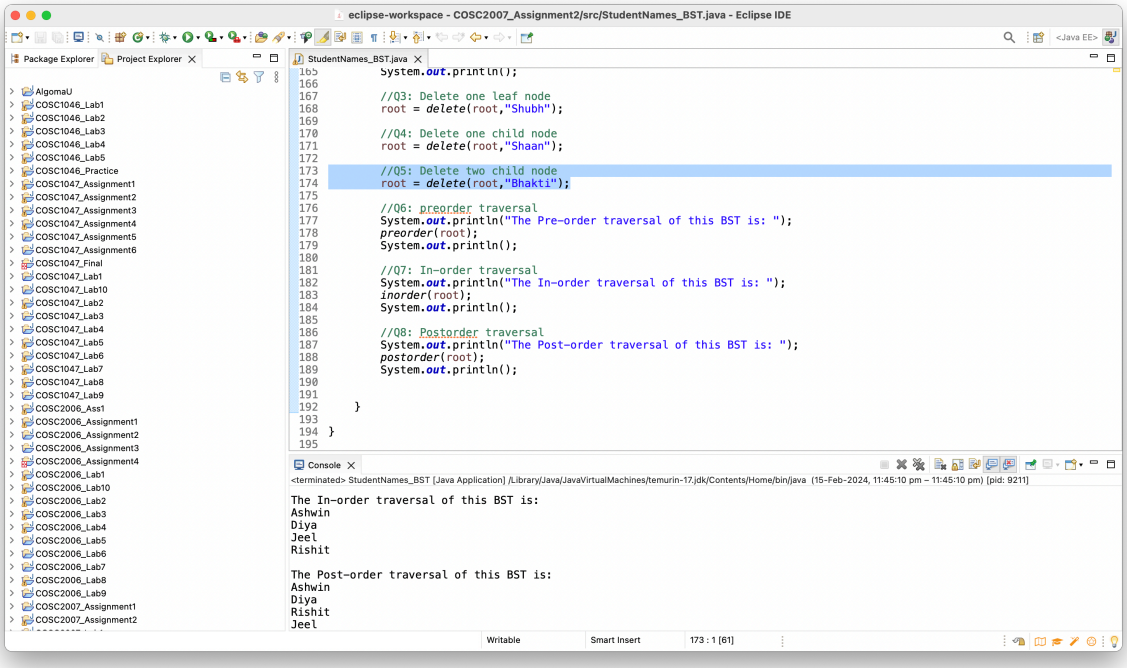
```
//main class
    public static void main(String[] args) {
```

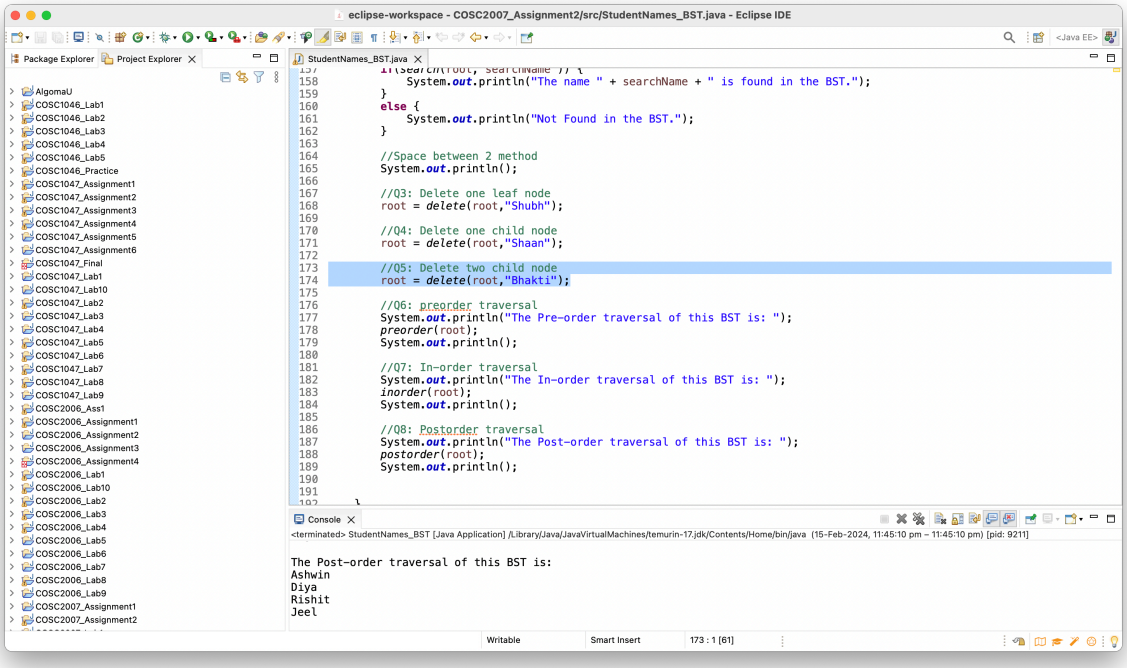
```
        // Q1: Construct a binary search tree
        String studentNames[] = {"Jeel", "Rishit",
"Bhakti", "Shubh", "Ashwin", "Shaan", "Diya"};
        Node root = null;

        //Multiple insert through for loop
        for(int i=0; i<studentNames.length; i++) {
            root = insert(root, studentNames[i]);
        }



        //Q2: Perform the search operation
        String searchName = "Harshil";
        if(search(root, searchName )) {
            System.out.println("The name " +
searchName + " is found in the BST.");
        }
        else {
            System.out.println("Not Found in the
BST.");
        }

        //Space between 2 method
        System.out.println();

        //Q3: Delete one leaf node
        root = delete(root,"Shubh");

        //Q4: Delete one child node
        root = delete(root,"Shaan");

        //Q5: Delete two child node
        root = delete(root,"Bhakti");

        //Q6: preorder traversal
        System.out.println("The Pre-order traversal
of this BST is: ");
        preorder(root);
        System.out.println();

        //Q7: In-order traversal
```

```
        System.out.println("The In-order traversal of
this BST is: ");
        inorder(root);
        System.out.println();

        //Q8: Postorder traversal
        System.out.println("The Post-order traversal
of this BST is: ");
        postorder(root);
        System.out.println();


    }

}
```

OUTPUT:







6 - Perform the pre-order traversal

Algorithm/ Pseudocode

//Preorder traversal

//check root first, then check left and then right node

Code

Preorder Method:

```
//Preorder traversal = root, left, right
```

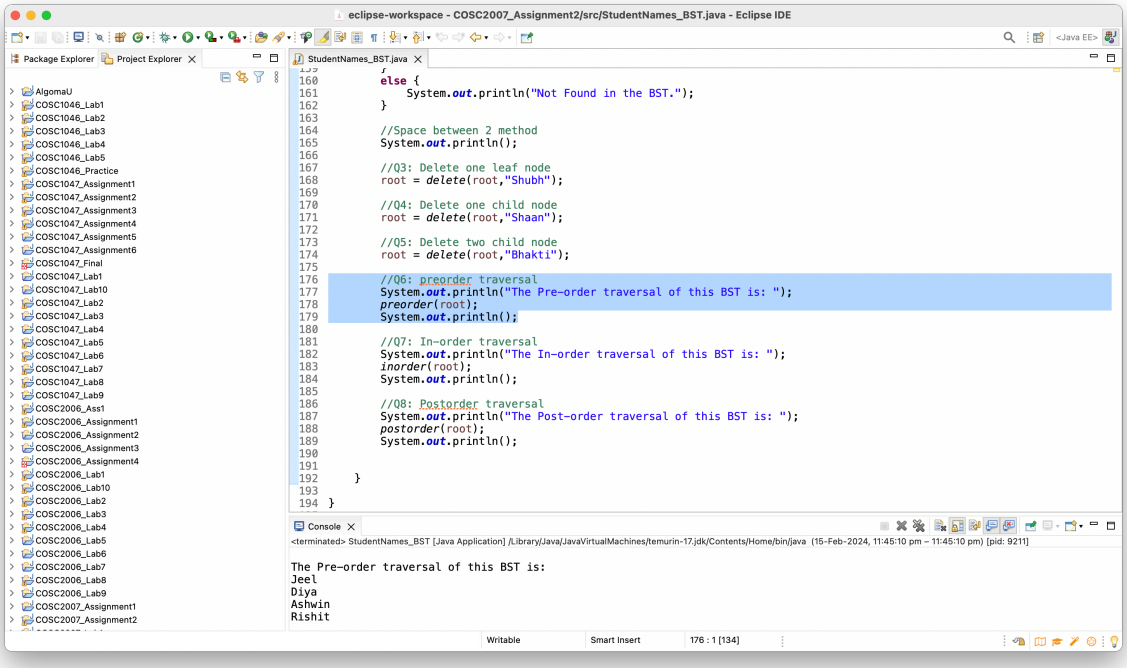
```
    public static void preorder(Node root) {
        if(root == null) {
            return;
        }

        System.out.println(root.data+ " ");
        preorder(root.left);
        preorder(root.right);

    }
```

MAIN METHOD:

```
//main class
    public static void main(String[] args) {

        // Q1: Construct a binary search tree
        String studentNames[] = {"Jeel", "Rishit",
"Bhakti", "Shubh", "Ashwin", "Shaan", "Diya"};
        Node root = null;

        //Multiple insert through for loop
        for(int i=0; i<studentNames.length; i++) {
            root = insert(root, studentNames[i]);
        }



        //Q2: Perform the search operation
        String searchName = "Harshil";
        if(search(root, searchName )) {
            System.out.println("The name " +
searchName + " is found in the BST.");
        }
        else {
            System.out.println("Not Found in the
BST.");
        }

        //Space between 2 method
        System.out.println();

        //Q3: Delete one leaf node
        root = delete(root,"Shubh");
```

```
        //Q4: Delete one child node
        root = delete(root,"Shaan");

        //Q5: Delete two child node
        root = delete(root,"Bhakti");

        //Q6: preorder traversal
        System.out.println("The Pre-order traversal
of this BST is: ");
        preorder(root);
        System.out.println();

        //Q7: In-order traversal
        System.out.println("The In-order traversal of
this BST is: ");
        inorder(root);
        System.out.println();

        //Q8: Postorder traversal
        System.out.println("The Post-order traversal
of this BST is: ");
        postorder(root);
        System.out.println();


    }

}
```

OUTPUT:



7 - Perform the in-order traversal

Algorithm/ Pseudocode

//check left node first
//then root
//then right node

Code:

In-order traversal method:

```
//inorder traversal = Left, root, right
public static void inorder(Node root){
    if(root == null) {
        return;
    }

    inorder(root.left);
    System.out.println(root.data+ " ");
    inorder(root.right);
}
```

MAIN METHOD:

```
//main class
    public static void main(String[] args) {

        // Q1: Construct a binary search tree
```

```
        String studentNames[] = {"Jeel", "Rishit",
"Bhakti", "Shubh", "Ashwin", "Shaan", "Diya"};
        Node root = null;

        //Multiple insert through for loop
        for(int i=0; i<studentNames.length; i++) {
            root = insert(root, studentNames[i]);
        }



        //Q2: Perform the search operation
        String searchName = "Harshil";
        if(search(root, searchName )) {
            System.out.println("The name " +
searchName + " is found in the BST.");
        }
        else {
            System.out.println("Not Found in the
BST.");
        }

        //Space between 2 method
        System.out.println();

        //Q3: Delete one leaf node
        root = delete(root,"Shubh");

        //Q4: Delete one child node
        root = delete(root,"Shaan");

        //Q5: Delete two child node
        root = delete(root,"Bhakti");

        //Q6: preorder traversal
        System.out.println("The Pre-order traversal
of this BST is: ");
        preorder(root);
        System.out.println();

        //Q7: In-order traversal
```

```
        System.out.println("The In-order traversal of
this BST is: ");
        inorder(root);
        System.out.println();

        //Q8: Postorder traversal
        System.out.println("The Post-order traversal
of this BST is: ");
        postorder(root);
        System.out.println();


    }

}
```

8 - Perform the post-order traversal

Algorithm/ Pseudocode

//check left node first
//check right node
//then at last, node

Code

Postorder method

```
//postorder traversal = Left, Right, root
public static void postorder(Node root) {
    if(root == null) {
        return;
    }

    postorder(root.left);
    postorder(root.right);
    System.out.println(root.data+ " ");

}
```

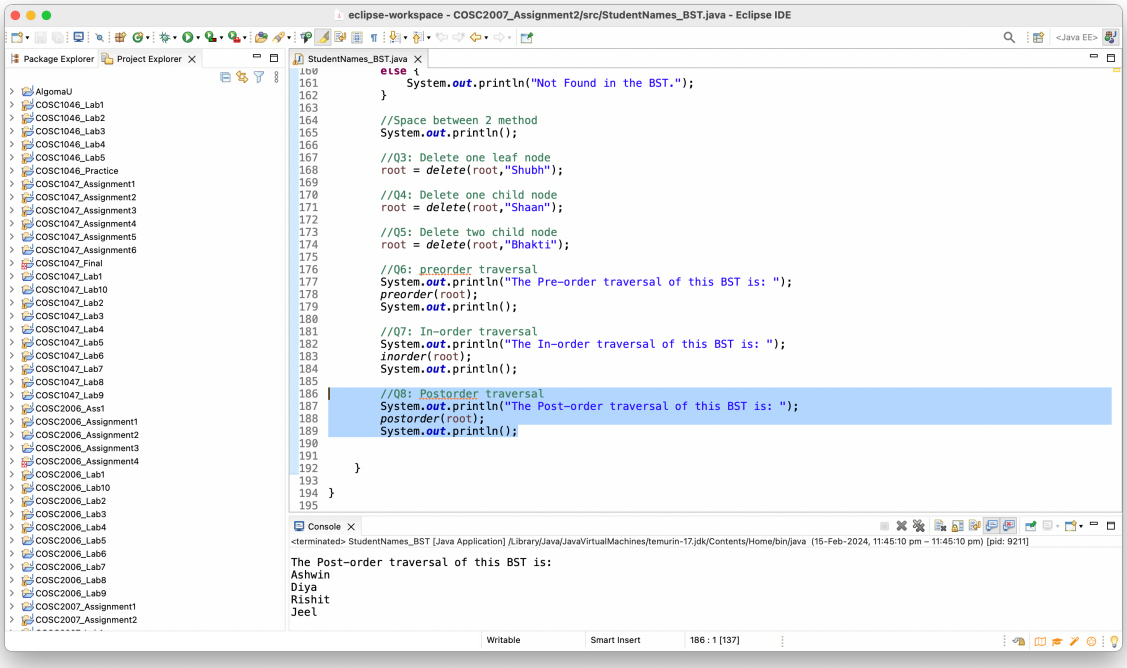
MAIN METHOD:

```
//main class
    public static void main(String[] args) {

        // Q1: Construct a binary search tree
```

```
        String studentNames[] = {"Jeel", "Rishit",
"Bhakti", "Shubh", "Ashwin", "Shaan", "Diya"};
        Node root = null;

        //Multiple insert through for loop
        for(int i=0; i<studentNames.length; i++) {
            root = insert(root, studentNames[i]);
        }



        //Q2: Perform the search operation
        String searchName = "Harshil";
        if(search(root, searchName )) {
            System.out.println("The name " +
searchName + " is found in the BST.");
        }
        else {
            System.out.println("Not Found in the
BST.");
        }

        //Space between 2 method
        System.out.println();

        //Q3: Delete one leaf node
        root = delete(root,"Shubh");

        //Q4: Delete one child node
        root = delete(root,"Shaan");

        //Q5: Delete two child node
        root = delete(root,"Bhakti");

        //Q6: preorder traversal
        System.out.println("The Pre-order traversal
of this BST is: ");
        preorder(root);
        System.out.println();

        //Q7: In-order traversal
```

```
        System.out.println("The In-order traversal of
this BST is: ");
        inorder(root);
        System.out.println();

        //Q8: Postorder traversal
        System.out.println("The Post-order traversal
of this BST is: ");
        postorder(root);
        System.out.println();


    }

}
```

OUTPUT:



CONCLUSION:

The above code is a Java program that implements a binary search tree (BST) of student names. The code defines a Node class that represents a node in the BST, and provides methods for inserting, searching, deleting, and traversing the tree. The main method creates a BST from an array of student names, and performs some operations on it, such as searching for a name, deleting some nodes, and printing the preorder, inorder, postorder traversal of the tree. The code demonstrates the basic concepts and operations of a BST data structure.