GTU Department Of Computer Engineering

CSE 222 / 505 – Spring 2023

Homework 5

Ahmet CEBECİ

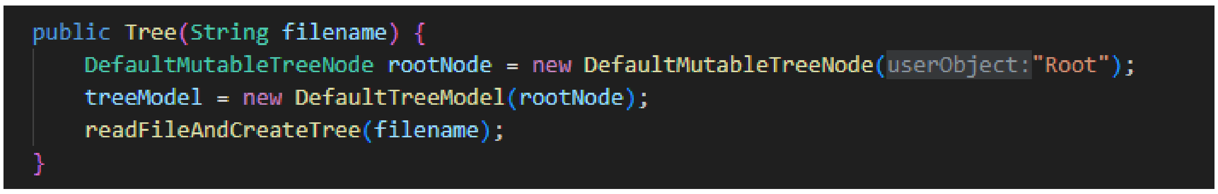
1901042708

CODE FUNCTIONALITY

1- class Tree

This class, named Tree, represents a tree data structure. It reads data from a file and constructs a tree from that data. The tree is stored as a DefaultTreeModel with DefaultMutableTreeNode as nodes.

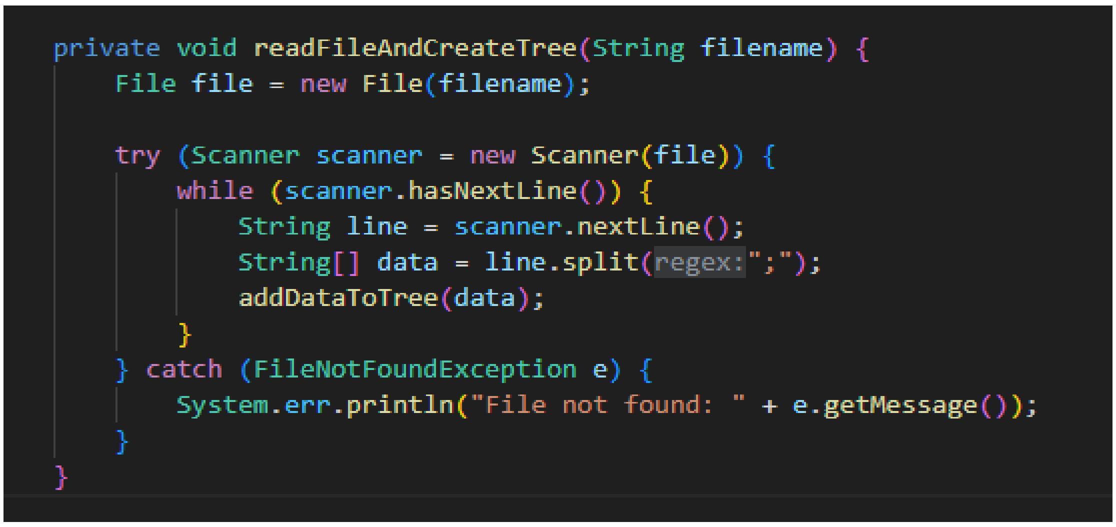
a-



```java
public Tree(String filename) {
    DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode(userObject:"Root");
    treeModel = new DefaultTreeModel(rootNode);
    readFileAndCreateTree(filename);
}
```

The constructor takes a filename as input and initializes the root node and the treeModel object. It then calls the readFileAndCreateTree() method.

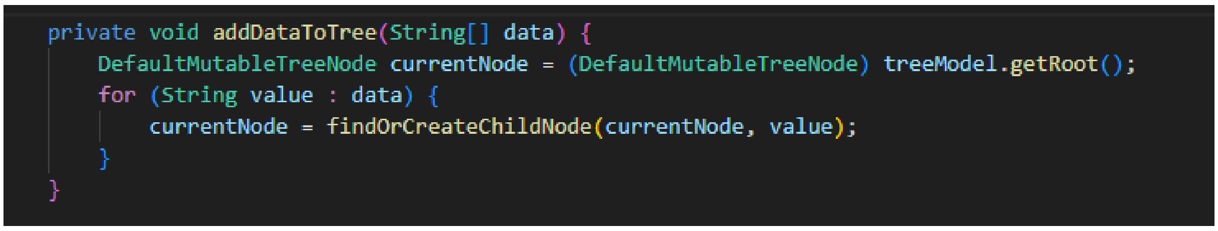
b-



```java
private void readFileAndCreateTree(String filename) {
    File file = new File(filename);

    try (Scanner scanner = new Scanner(file)) {
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] data = line.split(regex:";");
            addDataToTree(data);
        }
    } catch (FileNotFoundException e) {
        System.err.println("File not found: " + e.getMessage());
    }
}
```

This method reads the file line by line, splits each line into an array, and calls the addDataToTree() method to create the tree structure.

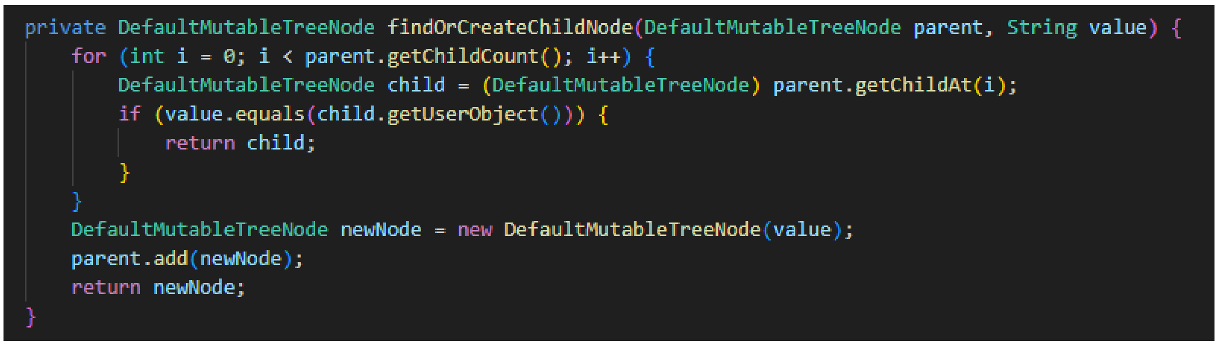
c-



```java
private void addDataToTree(String[] data) {
    DefaultMutableTreeNode currentNode = (DefaultMutableTreeNode) treeModel.getRoot();
    for (String value : data) {
        currentNode = findOrCreateChildNode(currentNode, value);
    }
}
```

Adds data to the tree by iterating through the data array and calling findOrCreateChildNode() for each value.

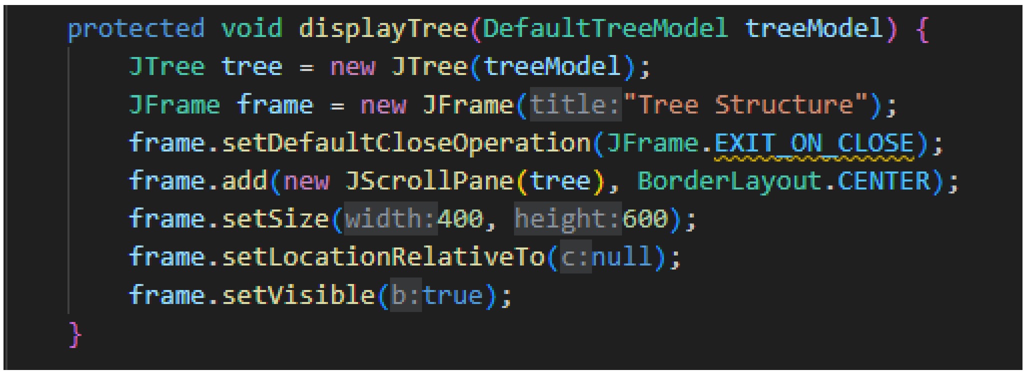
d-



```java
private DefaultMutableTreeNode findOrCreateChildNode(DefaultMutableTreeNode parent, String value) {
    for (int i = 0; i < parent.getChildCount(); i++) {
        DefaultMutableTreeNode child = (DefaultMutableTreeNode) parent.getChildAt(i);
        if (value.equals(child.getUserObject())) {
            return child;
        }
    }
    DefaultMutableTreeNode newNode = new DefaultMutableTreeNode(value);
    parent.add(newNode);
    return newNode;
}
```

This method checks if a node with the specified value exists among the children of the parent node. If not, it creates a new node with the specified value and adds it to the parent node.

e-



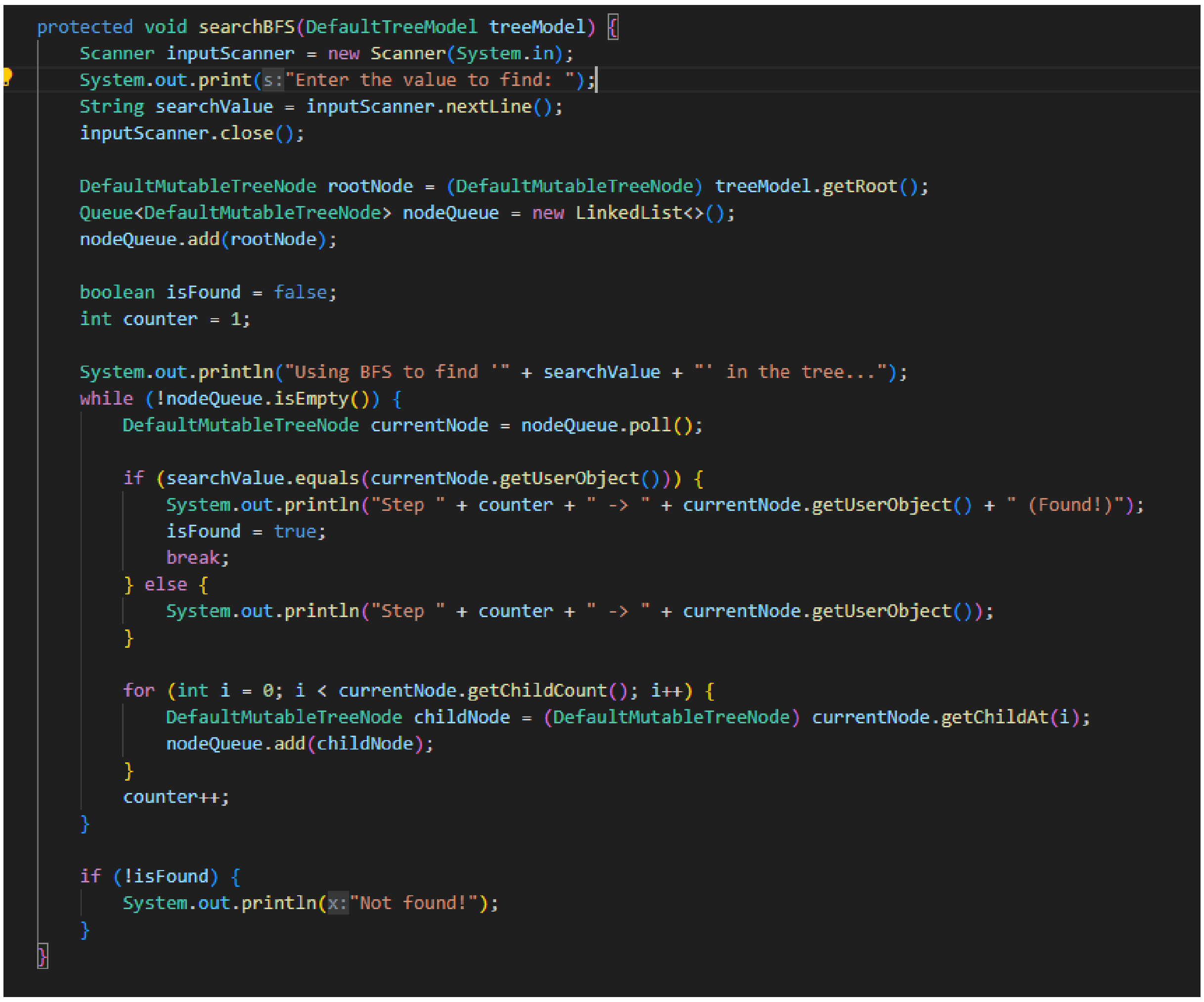
```java
protected void displayTree(DefaultTreeModel treeModel) {
    JTree tree = new JTree(treeModel);
    JFrame frame = new JFrame(title:"Tree Structure");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.add(new JScrollPane(tree), BorderLayout.CENTER);
    frame.setSize(width:400, height:600);
    frame.setLocationRelativeTo(c:null);
    frame.setVisible(b:true);
}
```

This method, called displayTree, accepts a DefaultTreeModel object as input and is designed to display a tree structure. It starts by creating a JTree object with the input tree model, which represents the tree structure. Next, a JFrame object, serving as the main window of the application, is created with the title "Tree Structure."

The method sets the default close operation for the window, so that when the user closes it, the application will terminate. To make the tree structure scrollable, a JScrollPane object is created and added to the JFrame, with the JTree object as its content. The scroll pane takes up the available space in the center of the JFrame by using BorderLayout.CENTER.

Finally, the method sets the size of the JFrame to 400 pixels wide and 600 pixels tall and positions it relative to the center of the screen. The window is then made visible, displaying the tree structure.

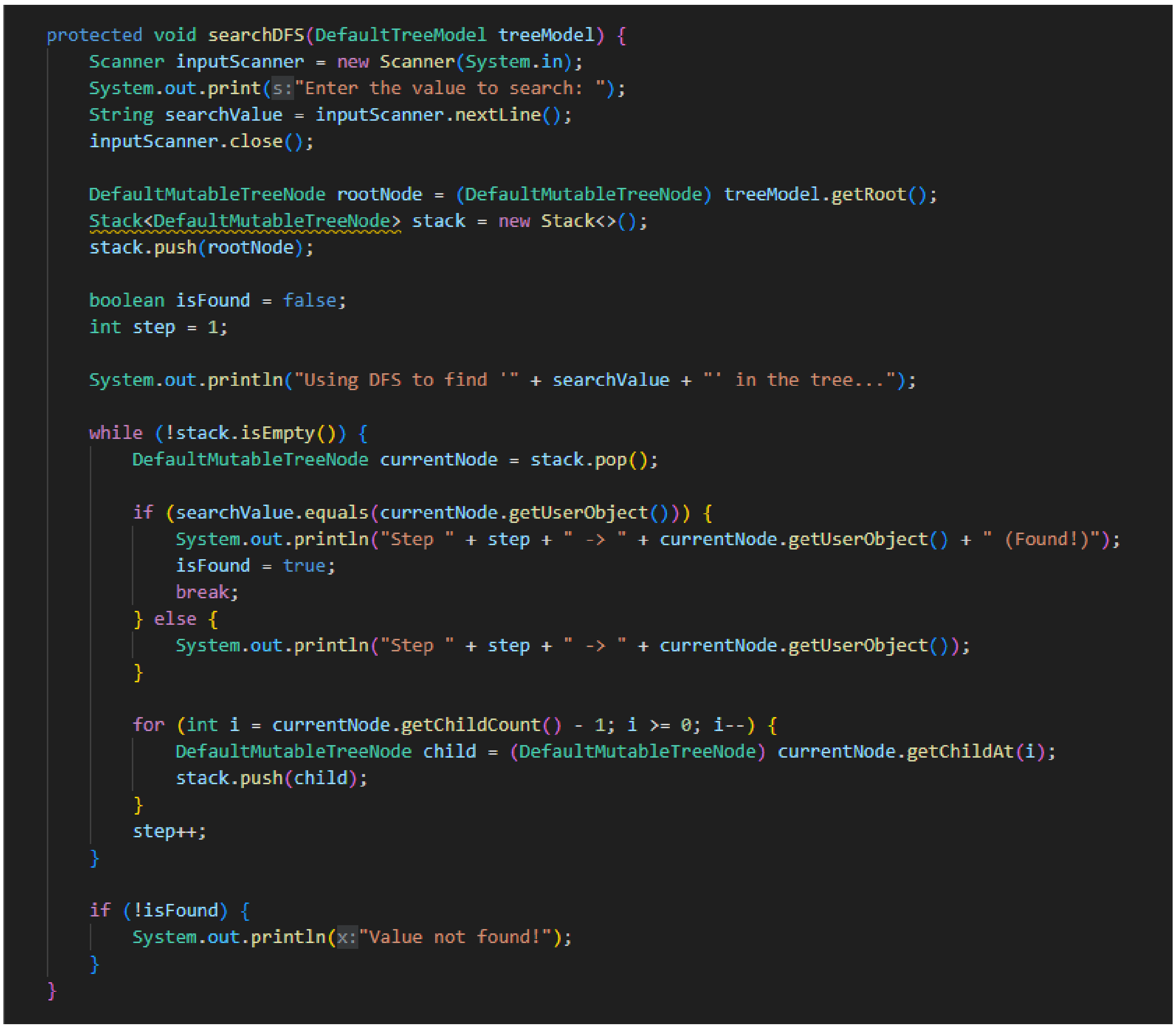
f-



```java
protected void searchBFS(DefaultTreeModel treeModel) {
    Scanner inputScanner = new Scanner(System.in);
    System.out.print(s:"Enter the value to find: ");
    String searchValue = inputScanner.nextLine();
    inputScanner.close();

    DefaultMutableTreeNode rootNode = (DefaultMutableTreeNode) treeModel.getRoot();
    Queue<DefaultMutableTreeNode> nodeQueue = new LinkedList<>();
    nodeQueue.add(rootNode);

    boolean isFound = false;
    int counter = 1;

    System.out.println("Using BFS to find '" + searchValue + "' in the tree...");
    while (!nodeQueue.isEmpty()) {
        DefaultMutableTreeNode currentNode = nodeQueue.poll();

        if (searchValue.equals(currentNode.getUserObject())) {
            System.out.println("Step " + counter + " -> " + currentNode.getUserObject() + " (Found!)");
            isFound = true;
            break;
        } else {
            System.out.println("Step " + counter + " -> " + currentNode.getUserObject());
        }

        for (int i = 0; i < currentNode.getChildCount(); i++) {
            DefaultMutableTreeNode childNode = (DefaultMutableTreeNode) currentNode.getChildAt(i);
            nodeQueue.add(childNode);
        }
        counter++;
    }

    if (!isFound) {
        System.out.println(x:"Not found!");
    }
}
```

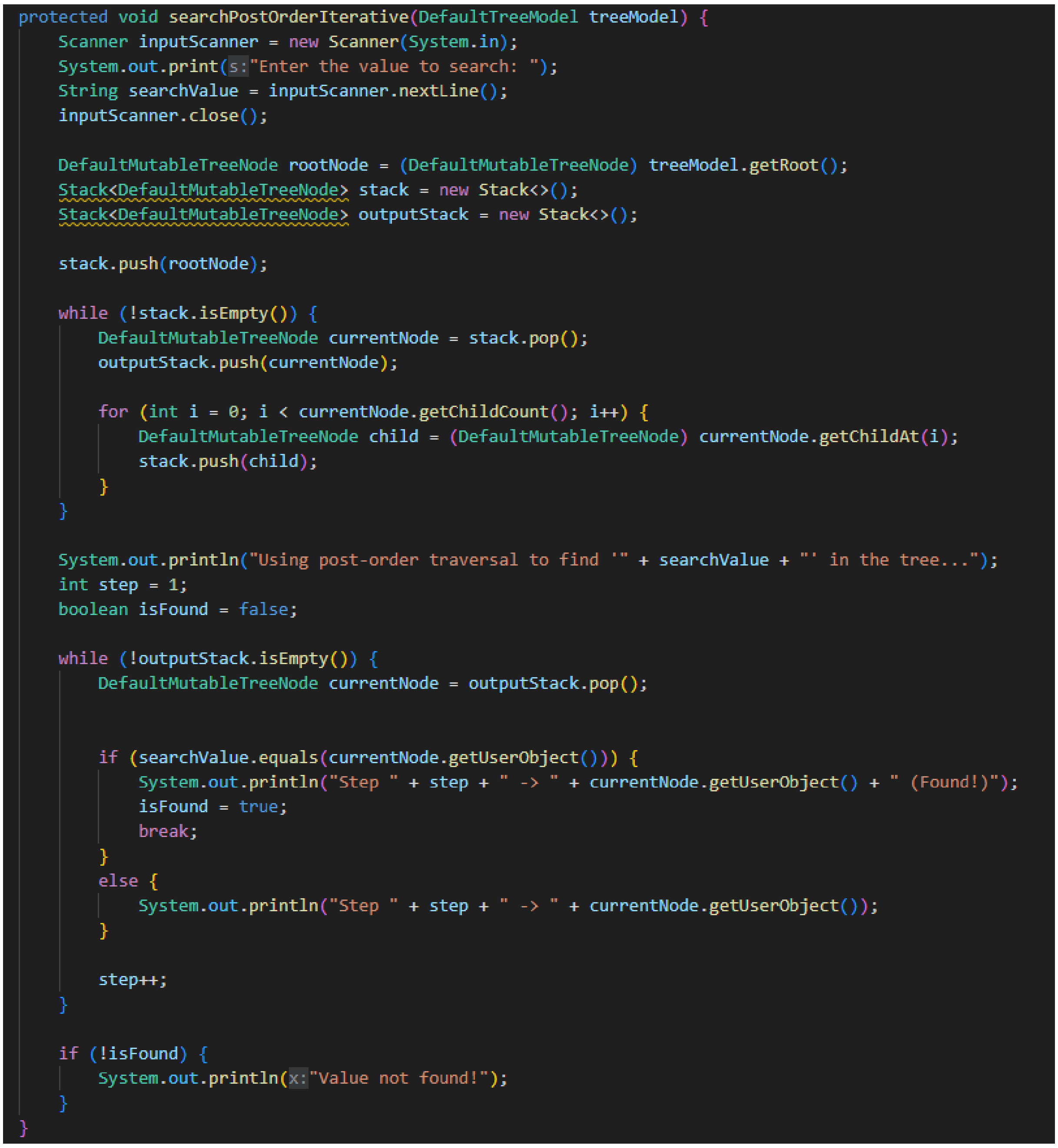
The searchBFS method performs a Breadth-First Search on a tree represented by a DefaultTreeModel. It takes user input for the value to find and initializes a queue with the tree's root node. Using a while loop, the method traverses the tree, comparing each node's value with the search value. If the value is found, a message is printed, and the loop is terminated. If the value is not found, the method continues traversing and enqueues child nodes until the queue is empty. If the value is not found after traversing the entire tree, a "Not found!" message is displayed.

g-



```java
protected void searchDFS(DefaultTreeModel treeModel) {
    Scanner inputScanner = new Scanner(System.in);
    System.out.print(s:"Enter the value to search: ");
    String searchValue = inputScanner.nextLine();
    inputScanner.close();

    DefaultMutableTreeNode rootNode = (DefaultMutableTreeNode) treeModel.getRoot();
    Stack<DefaultMutableTreeNode> stack = new Stack<>();
    stack.push(rootNode);

    boolean isFound = false;
    int step = 1;

    System.out.println("Using DFS to find '" + searchValue + "' in the tree...");

    while (!stack.isEmpty()) {
        DefaultMutableTreeNode currentNode = stack.pop();

        if (searchValue.equals(currentNode.getUserObject())) {
            System.out.println("Step " + step + " -> " + currentNode.getUserObject() + " (Found!)");
            isFound = true;
            break;
        } else {
            System.out.println("Step " + step + " -> " + currentNode.getUserObject());
        }

        for (int i = currentNode.getChildCount() - 1; i >= 0; i--) {
            DefaultMutableTreeNode child = (DefaultMutableTreeNode) currentNode.getChildAt(i);
            stack.push(child);
        }
        step++;
    }

    if (!isFound) {
        System.out.println(x:"Value not found!");
    }
}
```

The searchDFS method performs a Depth-First Search on a tree represented by a DefaultTreeModel. It takes user input for the value to search and initializes a stack with the tree's root node. Using a while loop, the method traverses the tree, comparing each node's value with the search value. If the value is found, a message is printed, and the loop is terminated. If the value is not found, the method continues traversing and pushes child nodes onto the stack until the stack is empty. If the value is not found after traversing the entire tree, a "Not found!" message is displayed i-



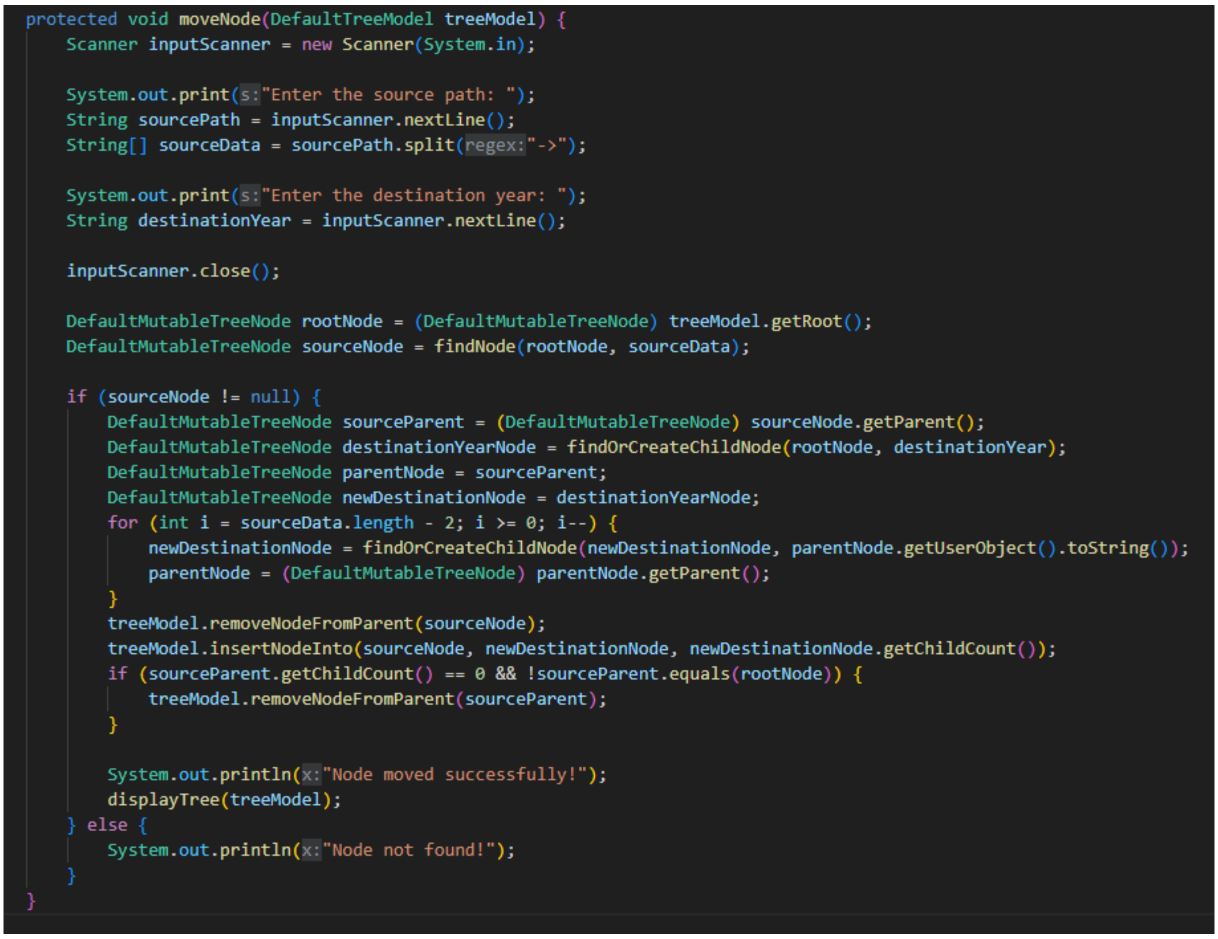
```java
protected void searchPostOrderIterative(DefaultTreeModel treeModel) {
    Scanner inputScanner = new Scanner(System.in);
    System.out.print(s:"Enter the value to search: ");
    String searchValue = inputScanner.nextLine();
    inputScanner.close();

    DefaultMutableTreeNode rootNode = (DefaultMutableTreeNode) treeModel.getRoot();
    Stack<DefaultMutableTreeNode> stack = new Stack<>();
    Stack<DefaultMutableTreeNode> outputStack = new Stack<>();

    stack.push(rootNode);

    while (!stack.isEmpty()) {
        DefaultMutableTreeNode currentNode = stack.pop();
        outputStack.push(currentNode);

        for (int i = 0; i < currentNode.getChildCount(); i++) {
            DefaultMutableTreeNode child = (DefaultMutableTreeNode) currentNode.getChildAt(i);
            stack.push(child);
        }
    }

    System.out.println("Using post-order traversal to find '" + searchValue + "' in the tree...");
    int step = 1;
    boolean isFound = false;

    while (!outputStack.isEmpty()) {
        DefaultMutableTreeNode currentNode = outputStack.pop();


        if (searchValue.equals(currentNode.getUserObject())) {
            System.out.println("Step " + step + " -> " + currentNode.getUserObject() + " (Found!)");
            isFound = true;
            break;
        }
        else {
            System.out.println("Step " + step + " -> " + currentNode.getUserObject());
        }

        step++;
    }

    if (!isFound) {
        System.out.println(x:"Value not found!");
    }
}
```

The searchPostOrderIterative method performs an iterative post-order traversal search on a tree represented by a DefaultTreeModel. It takes user input for the value to search and initializes two stacks: one for traversing the tree (stack) and another to store the post-order traversal result (outputStack).

The method pushes the root node onto the first stack and starts a while loop to traverse the tree. In each iteration, the current node is popped from the first stack and pushed onto the outputStack. Its child nodes are pushed onto the first stack in a left-to-right order.

After traversing the entire tree, the method uses another while loop to go through the outputStack, popping and comparing each node's value with the search value. If the value is found, a message is printed, and the loop is terminated. If the value is not found, the method continues traversing the outputStack until it is empty. If the value is not found after traversing the entire tree, a "Not found!" message is displayed.

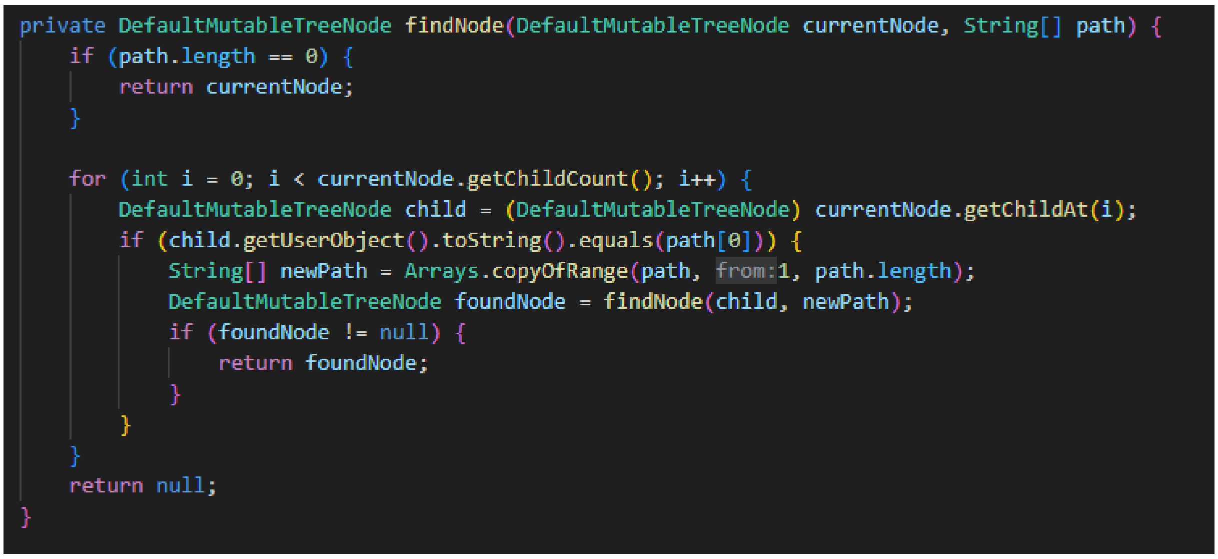
j-



```java
protected void moveNode(DefaultTreeModel treeModel) {
    Scanner inputScanner = new Scanner(System.in);

    System.out.print(s:"Enter the source path: ");
    String sourcePath = inputScanner.nextLine();
    String[] sourceData = sourcePath.split(regex:"->");

    System.out.print(s:"Enter the destination year: ");
    String destinationYear = inputScanner.nextLine();

    inputScanner.close();

    DefaultMutableTreeNode rootNode = (DefaultMutableTreeNode) treeModel.getRoot();
    DefaultMutableTreeNode sourceNode = findNode(rootNode, sourceData);

    if (sourceNode != null) {
        DefaultMutableTreeNode sourceParent = (DefaultMutableTreeNode) sourceNode.getParent();
        DefaultMutableTreeNode destinationYearNode = findOrCreateChildNode(rootNode, destinationYear);
        DefaultMutableTreeNode parentNode = sourceParent;
        DefaultMutableTreeNode newDestinationNode = destinationYearNode;
        for (int i = sourceData.length - 2; i >= 0; i--) {
            newDestinationNode = findOrCreateChildNode(newDestinationNode, parentNode.getUserObject().toString());
            parentNode = (DefaultMutableTreeNode) parentNode.getParent();
        }
        treeModel.removeNodeFromParent(sourceNode);
        treeModel.insertNodeInto(sourceNode, newDestinationNode, newDestinationNode.getChildCount());
        if (sourceParent.getChildCount() == 0 && !sourceParent.equals(rootNode)) {
            treeModel.removeNodeFromParent(sourceParent);
        }

        System.out.println(x:"Node moved successfully!");
        displayTree(treeModel);
    } else {
        System.out.println(x:"Node not found!");
    }
}
```

This code snippet defines a method called moveNode that takes a DefaultTreeModel as its argument. The method is responsible for moving a node from its original location to a new location within a tree structure. It first prompts the user to input the source path and destination year. After that, the method searches for the source node in the tree based on the user's input. If the source node is found, it is removed from its current parent and inserted into a new destination node. If the source node's parent is left with no children, it is also removed from the tree. Finally, the method prints out whether the node was moved successfully or not, and displays the updated tree.

k-

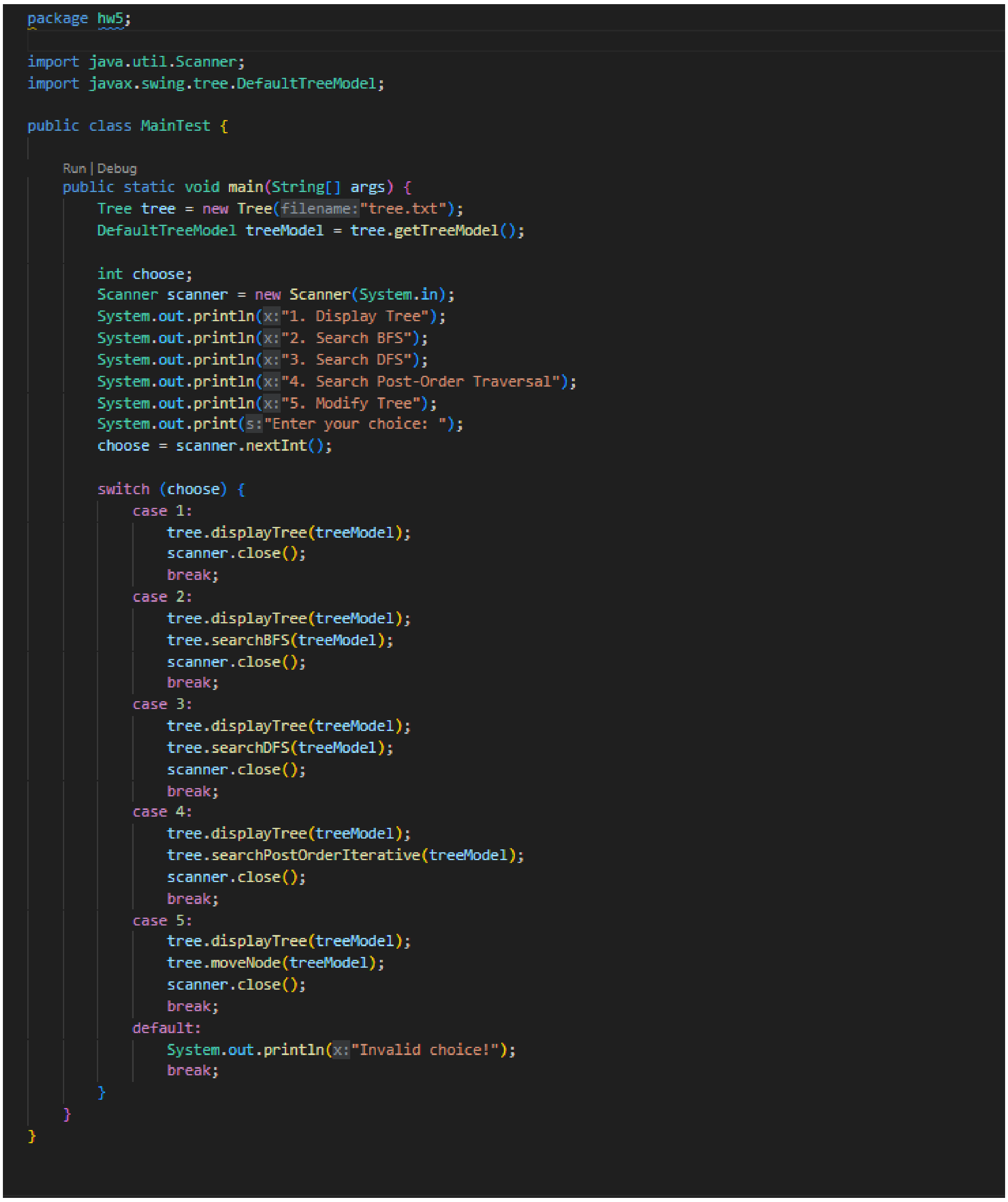


```java
private DefaultMutableTreeNode findNode(DefaultMutableTreeNode currentNode, String[] path) {
    if (path.length == 0) {
        return currentNode;
    }

    for (int i = 0; i < currentNode.getChildCount(); i++) {
        DefaultMutableTreeNode child = (DefaultMutableTreeNode) currentNode.getChildAt(i);
        if (child.getUserObject().toString().equals(path[0])) {
            String[] newPath = Arrays.copyOfRange(path, from:1, path.length);
            DefaultMutableTreeNode foundNode = findNode(child, newPath);
            if (foundNode != null) {
                return foundNode;
            }
        }
    }
    return null;
}
```

findNode, which takes a DefaultMutableTreeNode called currentNode and a String array called path. The method is a recursive function that searches for a node within a tree structure based on the given path.

If the path is empty, the method returns the currentNode as the desired node has been found. The method iterates through the children of the currentNode, and if a child node's user object matches the first element in the path array, it calls the findNode method recursively with the child node as the new currentNode and the remaining path elements. If the recursive call returns a non-null result, it means the node has been found, and the method returns that node. If the node is not found, the method returns null.

2- class MainTest



```java
package hw5;

import java.util.Scanner;
import javax.swing.tree.DefaultTreeModel;

public class MainTest {

    Run | Debug
    public static void main(String[] args) {
        Tree tree = new Tree(filename:"tree.txt");
        DefaultTreeModel treeModel = tree.getTreeModel();

        int choose;
        Scanner scanner = new Scanner(System.in);
        System.out.println(x:"1. Display Tree");
        System.out.println(x:"2. Search BFS");
        System.out.println(x:"3. Search DFS");
        System.out.println(x:"4. Search Post-Order Traversal");
        System.out.println(x:"5. Modify Tree");
        System.out.print(s:"Enter your choice: ");
        choose = scanner.nextInt();

        switch (choose) {
            case 1:
                tree.displayTree(treeModel);
                scanner.close();
                break;
            case 2:
                tree.displayTree(treeModel);
                tree.searchBFS(treeModel);
                scanner.close();
                break;
            case 3:
                tree.displayTree(treeModel);
                tree.searchDFS(treeModel);
                scanner.close();
                break;
            case 4:
                tree.displayTree(treeModel);
                tree.searchPostOrderIterative(treeModel);
                scanner.close();
                break;
            case 5:
                tree.displayTree(treeModel);
                tree.moveNode(treeModel);
                scanner.close();
                break;
            default:
                System.out.println(x:"Invalid choice!");
                break;
        }
    }
}
```

This MainTest class is a test driver for the Tree class. It creates an instance of the Tree class by reading data from a file named "tree.txt" and retrieves the tree model. It then displays a menu to the user to choose from various operations:

1-Display Tree

2-Search BFS (Breadth-First Search)

3-Search DFS (Depth-First Search)

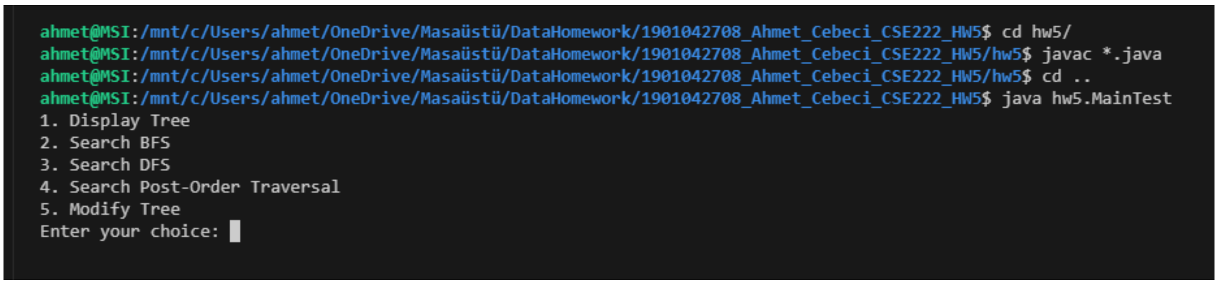
4-Search Post-Order Traversal

5-Modify Tree

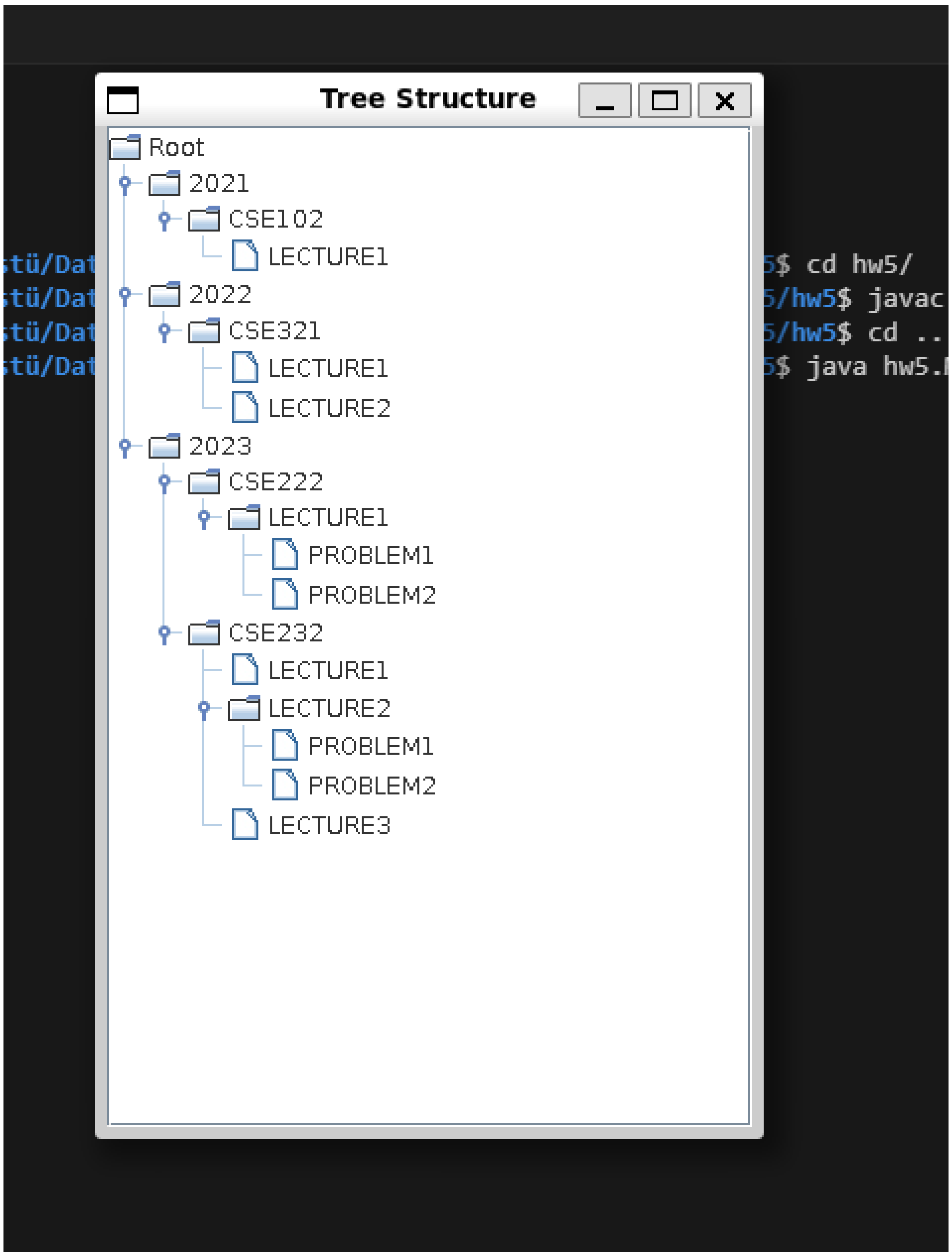
The user inputs their choice, and the program performs the corresponding operation using the methods from the Tree class. The switch statement is used to call the appropriate method based on the user's choice. The scanner.close() method is called after the chosen operation is completed to close the scanner and release its resources.

To run this program, make sure you have a file named "tree.txt" with the appropriate data in the same directory as your MainTest class. The data should be formatted with each line representing a path in the tree, with values separated by semicolons.

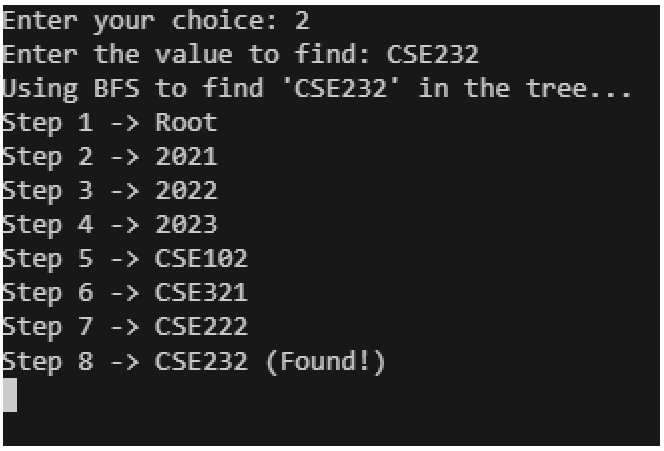
OUTPUT



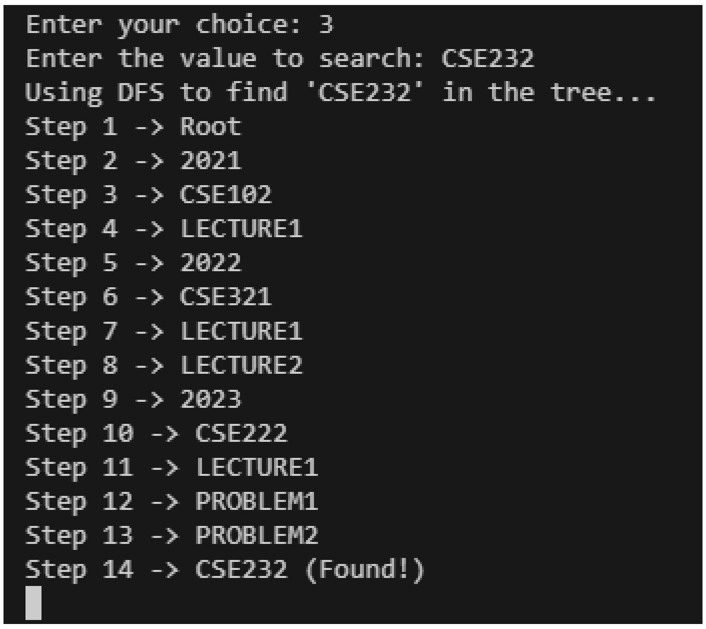
```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/1901042708_Ahmet_Cebeci_CSE222_HW5$ cd hw5/
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/1901042708_Ahmet_Cebeci_CSE222_HW5/hw5$ javac *.java
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/1901042708_Ahmet_Cebeci_CSE222_HW5/hw5$ cd ..
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/1901042708_Ahmet_Cebeci_CSE222_HW5$ java hw5.MainTest
1. Display Tree
2. Search BFS
3. Search DFS
4. Search Post-Order Traversal
5. Modify Tree
Enter your choice:
```



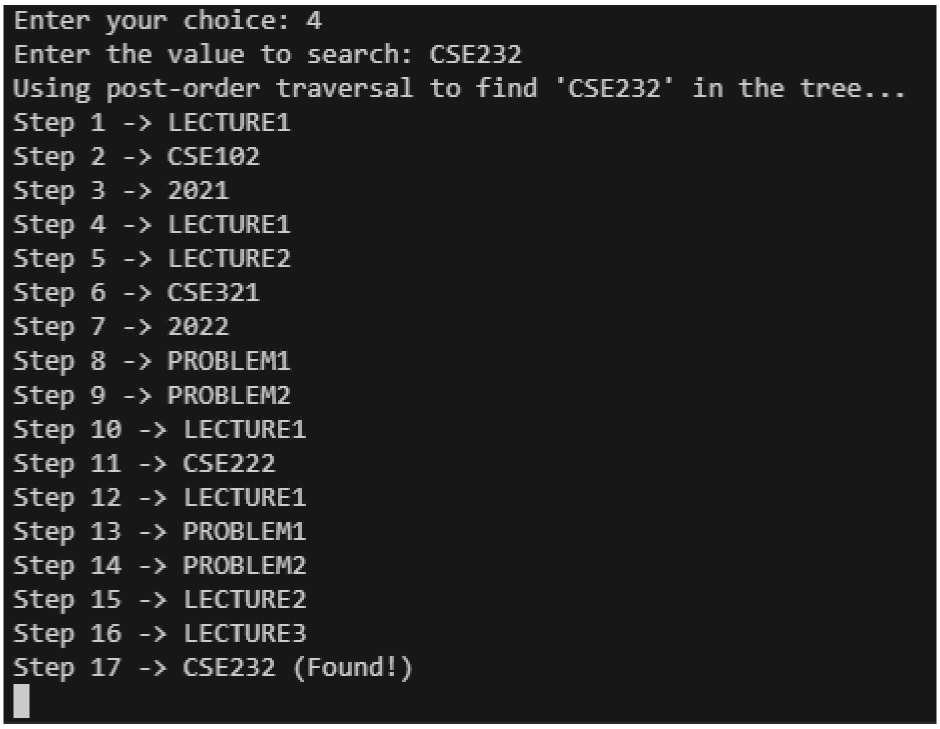
```
Tree Structure
Root
  2021
    CSE102
      LECTURE1
  2022
    CSE321
      LECTURE1
      LECTURE2
  2023
    CSE222
      LECTURE1
        PROBLEM1
        PROBLEM2
    CSE232
      LECTURE1
      LECTURE2
        PROBLEM1
        PROBLEM2
      LECTURE3
```



```
Enter your choice: 2
Enter the value to find: CSE232
Using BFS to find 'CSE232' in the tree...
Step 1 -> Root
Step 2 -> 2021
Step 3 -> 2022
Step 4 -> 2023
Step 5 -> CSE102
Step 6 -> CSE321
Step 7 -> CSE222
Step 8 -> CSE232 (Found!)
```



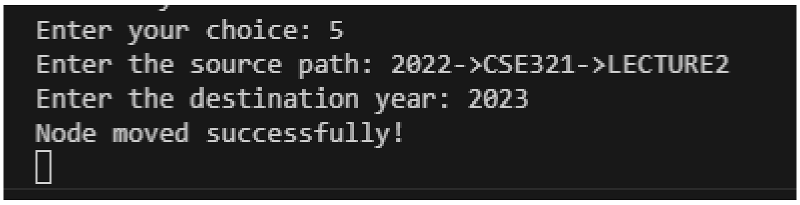
```
Enter your choice: 3
Enter the value to search: CSE232
Using DFS to find 'CSE232' in the tree...
Step 1 -> Root
Step 2 -> 2021
Step 3 -> CSE102
Step 4 -> LECTURE1
Step 5 -> 2022
Step 6 -> CSE321
Step 7 -> LECTURE1
Step 8 -> LECTURE2
Step 9 -> 2023
Step 10 -> CSE222
Step 11 -> LECTURE1
Step 12 -> PROBLEM1
Step 13 -> PROBLEM2
Step 14 -> CSE232 (Found!)
```



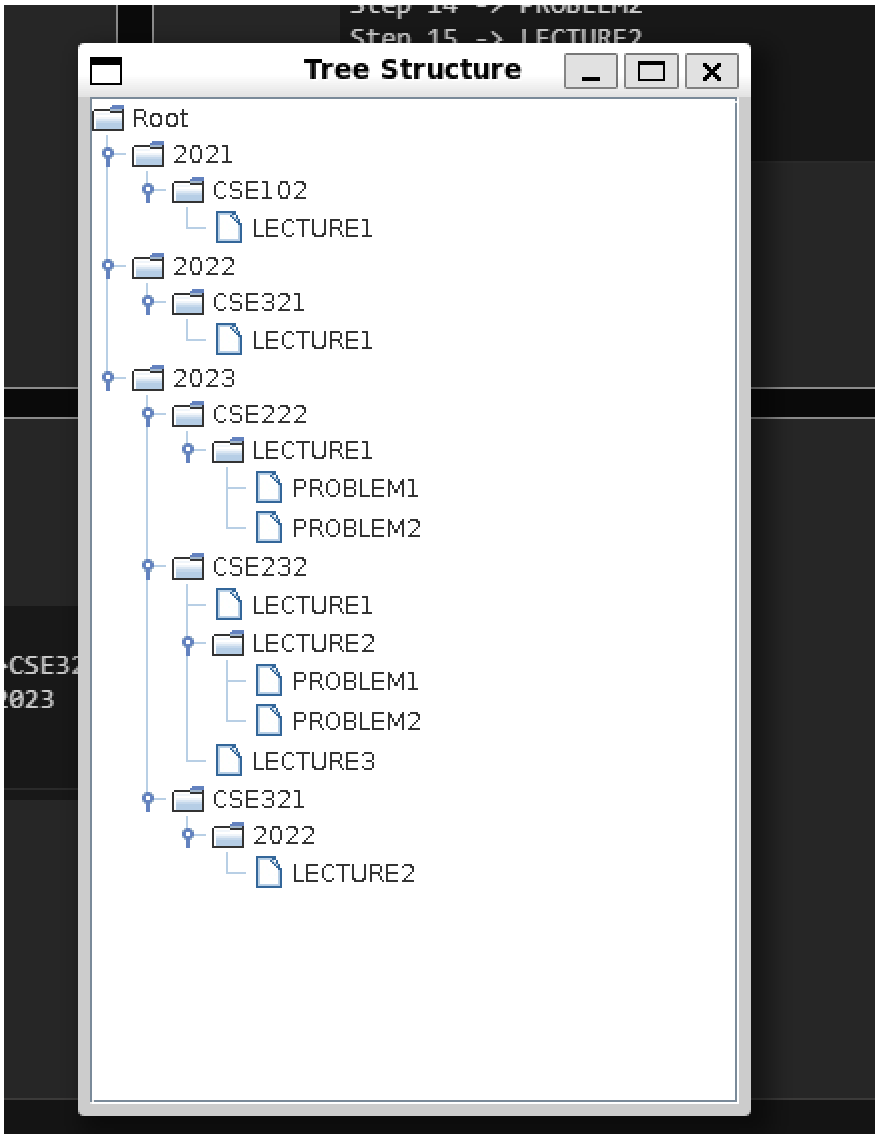
```
Enter your choice: 4
Enter the value to search: CSE232
Using post-order traversal to find 'CSE232' in the tree...
Step 1 -> LECTURE1
Step 2 -> CSE102
Step 3 -> 2021
Step 4 -> LECTURE1
Step 5 -> LECTURE2
Step 6 -> CSE321
Step 7 -> 2022
Step 8 -> PROBLEM1
Step 9 -> PROBLEM2
Step 10 -> LECTURE1
Step 11 -> CSE222
Step 12 -> LECTURE1
Step 13 -> PROBLEM1
Step 14 -> PROBLEM2
Step 15 -> LECTURE2
Step 16 -> LECTURE3
Step 17 -> CSE232 (Found!)
```

PART E)

NODE MOVE

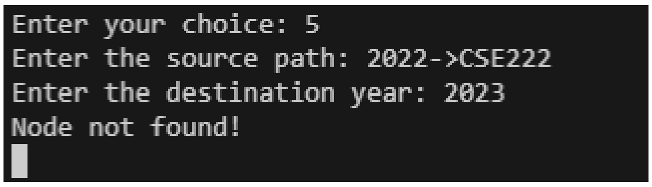


```
Enter your choice: 5
Enter the source path: 2022->CSE321->LECTURE2
Enter the destination year: 2023
Node moved successfully!
```



```
Tree Structure
Root
  2021
    CSE102
      LECTURE1
  2022
    CSE321
      LECTURE1
  2023
    CSE222
      LECTURE1
        PROBLEM1
        PROBLEM2
    CSE232
      LECTURE1
      LECTURE2
        PROBLEM1
        PROBLEM2
      LECTURE3
    CSE321
      2022
        LECTURE2
```

NODE NOT FOUND



```
Enter your choice: 5
Enter the source path: 2022->CSE222
Enter the destination year: 2023
Node not found!
```