import java.util.\*;

import java.io.\*;

class Info {

    private String data;

    private int number;

    public String getData() {

        return data;

    }

    public void setData(String data) {

        this.data = data;

    }

    public int getNumber() {

        return number;

    }

    public void setNumber(int number) {

        this.number = number;

    }

    @Override

    public String toString() {

        return "(" + data + ", " + number + ")";

    }

}

class Node {

    private Info info;

    private Node left;

    private Node right;

    public Info getInfo() {

        return info;

    }

    public void setInfo(Info info) {

        this.info = info;

    }

    public Node getLeft() {

        return left;

    }

    public void setLeft(Node left) {

        this.left = left;

    }

    public Node getRight() {

        return right;

    }

    public void setRight(Node right) {

        this.right = right;

    }

}

class BinaryTree {

    private Node root;

    public void insert(Info info) {

        root = insertRecursive(root, info);

    }

    private Node insertRecursive(Node root, Info info) {

        if (root == null) {

            Node newNode = new Node();

            newNode.setInfo(info);

            return newNode;

        }

        if (info.getNumber() < root.getInfo().getNumber()) {

            root.setLeft(insertRecursive(root.getLeft(), info));

        } else if (info.getNumber() > root.getInfo().getNumber()) {

            root.setRight(insertRecursive(root.getRight(), info));

        }

        return root;

    }

    public void printPreorder() {

        printPreorderRec(root);

    }

    private void printPreorderRec(Node root) {

        if (root != null) {

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

            printPreorderRec(root.getLeft());

            printPreorderRec(root.getRight());

        }

    }

    public void printPostorder() {

        printPostorderRec(root);

    }

    private void printPostorderRec(Node root) {

        if (root != null) {

            printPostorderRec(root.getLeft());

            printPostorderRec(root.getRight());

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

        }

    }

    public void printInorder() {

        printInorderRec(root);

    }

    private void printInorderRec(Node root) {

        if (root != null) {

            printInorderRec(root.getLeft());

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

            printInorderRec(root.getRight());

        }

    }

    public int count() {

        return countNodes(root, 1); // Start counting from level 1

    }

    private int countNodes(Node root, int level) {

        if (root == null)

            return 0;

        else

            return 1 + countNodes(root.getLeft(), level + 1) + countNodes(root.getRight(), level + 1);

    }

    public Info search(int num) {

        return searchRec(root, num);

    }

    private Info searchRec(Node root, int num) {

        if (root == null || root.getInfo().getNumber() == num) {

            if (root != null) {

                return root.getInfo();

            } else {

                return null;

            }

        }

        if (num < root.getInfo().getNumber()) {

            return searchRec(root.getLeft(), num);

        } else {

            return searchRec(root.getRight(), num);

        }

    }

    public void delete(int num) {

        root = deleteRec(root, num);

    }

    private Node deleteRec(Node root, int num) {

        if (root == null)

            return root;

        if (num < root.getInfo().getNumber())

            root.setLeft(deleteRec(root.getLeft(), num));

        else if (num > root.getInfo().getNumber())

            root.setRight(deleteRec(root.getRight(), num));

        else {

            if (root.getLeft() == null)

                return root.getRight();

            else if (root.getRight() == null)

                return root.getLeft();

            root.setInfo(minValueNode(root.getRight()).getInfo());

            root.setRight(deleteRec(root.getRight(), root.getInfo().getNumber()));

        }

        return root;

    }

    private Node minValueNode(Node node) {

        Node current = node;

        while (current.getLeft() != null)

            current = current.getLeft();

        return current;

    }

    public void edit(int oldNum, Info newInfo) {

        delete(oldNum);

        insert(newInfo);

    }

}

public class Main {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        BinaryTree tree = new BinaryTree();

        int choice;

        do {

            System.out.println("Menu");

            System.out.println("1) Insert");

            System.out.println("2) PrintPreorder");

            System.out.println("3) PrintPostorder");

            System.out.println("4) PrintInorder");

            System.out.println("5) Count");

            System.out.println("6) Search");

            System.out.println("7) Delete");

            System.out.println("8) Edit");

            System.out.println("9) Exit");

            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();

            scanner.nextLine(); // Consume newline

            switch (choice) {

                case 1:

                try {

                    File numberFile = new File("Numbers");

                    Scanner numberScanner = new Scanner(numberFile);

                    File infoFile = new File("Names");

                    Scanner infoScanner = new Scanner(infoFile);

                    while (numberScanner.hasNextLine() && infoScanner.hasNextLine()) {

                        int number = Integer.parseInt(numberScanner.nextLine());

                        String data = infoScanner.nextLine();

                        Info info = new Info();

                        System.out.println(number + " " + data);

                        info.setData(data);

                        info.setNumber(number);

                        tree.insert(info);

                    }

                    numberScanner.close();

                    infoScanner.close();

                } catch (FileNotFoundException e) {

                    System.err.println("File not found: " + e.getMessage());

                }

                break;

                case 2:

                    System.out.println("Preorder traversal:");

                    tree.printPreorder();

                    System.out.println();

                    break;

                case 3:

                    System.out.println("Postorder traversal:");

                    tree.printPostorder();

                    System.out.println();

                    break;

                case 4:

                    System.out.println("Inorder traversal:");

                    tree.printInorder();

                    System.out.println();

                    break;

                case 5:

                    System.out.println("Number of nodes in the tree: " + tree.count());

                    break;

                case 6:

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

                    int numToSearch = scanner.nextInt();

                    Info result = tree.search(numToSearch);

                    if (result != null) {

                        System.out.println("Search for Number " + numToSearch + ": Found - " + result);

                    } else {

                        System.out.println("Search for Number " + numToSearch + ": Not Found");

                        }

                    break;

                case 7:

                    System.out.print("Enter Number to delete: ");

                    int numToDelete = scanner.nextInt();

                    tree.delete(numToDelete);

                    System.out.println("Inorder traversal after deletion:");

                    tree.printInorder();

                    System.out.println();

                    break;

                case 8:

                    Info newInfo = new Info();

                    System.out.print("Enter old number: ");

                    int oldNum =

                    scanner.nextInt();

                    scanner.nextLine(); // Consume newline

                    System.out.print("Enter new data: ");

                    newInfo.setData(scanner.nextLine());

                    System.out.print("Enter new number: ");

                    newInfo.setNumber(scanner.nextInt());

                    tree.edit(oldNum, newInfo);

                    System.out.println("Inorder traversal after editing:");

                    tree.printInorder();

                    System.out.println();

                    break;

                case 9:

                    System.out.println("exit");

                    break;

                default:

                    System.out.println("no");

            }

        } while (choice != 9);

        scanner.close();

    }

}

**A computer screen shot of a black screen

Description automatically generated  
A computer screen shot of a black screen

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated**