mport java.util.ArrayList;

import java.util.Scanner;

// Contact class to store name, phone, email

class Contact {

    private String name;

    private String phoneNumber;

    private String email;

    public Contact(String name, String phoneNumber, String email) {

        this.name = name;

        this.phoneNumber = phoneNumber;

        this.email = email;

    }

    // Getters

    public String getName() {

        return name;

    }

    public String getPhoneNumber() {

        return phoneNumber;

    }

    public String getEmail() {

        return email;

    }

    // Display contact info

    public void displayContact() {

        System.out.println("Name: " + name);

        System.out.println("Phone Number: " + phoneNumber);

        System.out.println("Email: " + email);

        System.out.println("----------------------");

    }

}

// Main application class

public class PhoneBookApp {

    private static ArrayList<Contact> contacts = new ArrayList<>();

    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {

        int choice;

        do {

            System.out.println("\nPhonebook Menu:");

            System.out.println("1. Add Contact");

            System.out.println("2. View All Contacts");

            System.out.println("3. Search Contact by Name");

            System.out.println("4. Delete Contact");

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

            System.out.print("Choose an option: ");

            choice = scanner.nextInt();

            scanner.nextLine();  // Consume newline

            switch (choice) {

                case 1:

                    addContact();

                    break;

                case 2:

                    viewContacts();

                    break;

                case 3:

                    searchContact();

                    break;

                case 4:

                    deleteContact();

                    break;

                case 5:

                    System.out.println("Exiting Phonebook. Goodbye!");

                    break;

                default:

                    System.out.println("Invalid option. Please try again.");

            }

        } while (choice != 5);

    }

    // Add a new contact

    private static void addContact() {

        System.out.print("Enter Name: ");

        String name = scanner.nextLine();

        System.out.print("Enter Phone Number: ");

        String phoneNumber = scanner.nextLine();

        System.out.print("Enter Email: ");

        String email = scanner.nextLine();

        contacts.add(new Contact(name, phoneNumber, email));

        System.out.println("Contact added successfully.");

    }

    // View all contacts

    private static void viewContacts() {

        if (contacts.isEmpty()) {

            System.out.println("Phonebook is empty.");

            return;

        }

        System.out.println("\nAll Contacts:");

        for (Contact c : contacts) {

            c.displayContact();

        }

    }

    // Search for a contact by name

    private static void searchContact() {

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

        String searchName = scanner.nextLine();

        boolean found = false;

        for (Contact c : contacts) {

            if (c.getName().equalsIgnoreCase(searchName)) {

                System.out.println("Contact Found:");

                c.displayContact();

                found = true;

                break;

            }

        }

        if (!found) {

            System.out.println("No contact found with name: " + searchName);

        }

    }

    // Delete a contact by name

    private static void deleteContact() {

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

        String nameToDelete = scanner.nextLine();

        boolean removed = contacts.removeIf(contact -> contact.getName().equalsIgnoreCase(nameToDelete));

        if (removed) {

            System.out.println("Contact deleted successfully.");

        } else {

            System.out.println("Contact not found.");

        }

    }

}

BubbleSortExample:

class Person {

    String name;

    String phoneNumber;

    String email;

    // Constructor to initialize the fields

    public Person(String name, String phoneNumber, String email) {

        this.name = name;

        this.phoneNumber = phoneNumber;

        this.email = email;

    }

    // Method to display person's information

    public void display() {

        System.out.println("Name: " + name + ", Phone: " + phoneNumber + ", Email: " + email);

    }

}

public class BubbleSortExample {

    // Bubble sort method to sort array of Person objects by name

    public static void bubbleSort(Person[] people) {

        int n = people.length;

        Person temp;

        // Outer loop for passes

        for (int i = 0; i < n - 1; i++) {

            // Inner loop for comparisons in each pass

            for (int j = 0; j < n - 1 - i; j++) {

                // Compare names lexicographically

                if (people[j].name.compareToIgnoreCase(people[j + 1].name) > 0) {

                    // Swap if current name is greater than next name

                    temp = people[j];

                    people[j] = people[j + 1];

                    people[j + 1] = temp;

                }

            }

        }

    }

    public static void main(String[] args) {

        // Create an array of Person objects

        Person[] people = {

            new Person("Alice", "123-456-7890", "alice@example.com"),

            new Person("Charlie", "555-666-7777", "charlie@example.com"),

            new Person("Bob", "987-654-3210", "bob@example.com"),

            new Person("David", "111-222-3333", "david@example.com")

        };

        System.out.println("Before sorting:");

        for (Person p : people) {

            p.display();

        }

        // Sort the array by name

        bubbleSort(people);

        System.out.println("\nAfter sorting by name:");

        for (Person p : people) {

            p.display();

        }

    }

}

INSERTONSORTDESC:

public class InsertionSortDescending {

    public static void insertionSortDesc(int[] arr) {

        int n = arr.length;

        for (int i = 1; i < n; i++) {

            int key = arr[i];

            int j = i - 1;

            // Move elements of arr[0..i-1] that are smaller than key

            // to one position ahead of their current position

            while (j >= 0 && arr[j] < key) {

                arr[j + 1] = arr[j];

                j--;

            }

            arr[j + 1] = key;

        }

    }

    public static void main(String[] args) {

        int[] arr = {5, 1, 4, 2, 8};

        insertionSortDesc(arr);

        System.out.println("Sorted array in descending order:");

        for (int num : arr) {

            System.out.print(num + " ");

        }

    }

}

SORTSTRINGS:

import java.util.Arrays;

public class SortStrings {

    public static void main(String[] args) {

        String[] words = {"banana", "apple", "cherry", "date", "elderberry"};

        // Sort the array alphabetically

        Arrays.sort(words);

        // Print the sorted array

        System.out.println("Sorted strings:");

        for (String word : words) {

            System.out.println(word);

        }

    }

}

COUNTSWAPS:

public class CountSwaps {

    public static int countSwaps(int[] arr) {

        int n = arr.length;

        int swapCount = 0;

        for (int i = 0; i < n - 1; i++) {

            for (int j = 0; j < n - 1; j++) {

                if (arr[j] > arr[j + 1]) {

                    // Swap elements

                    int temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;

                    swapCount++;

                }

            }

        }

        return swapCount;

    }

    public static void main(String[] args) {

        int[] arr = {5, 1, 4, 2};

        int swaps = countSwaps(arr);

        System.out.println("Total Swaps: " + swaps);

    }

}

OPTIMIZED BUBBLE SORT:

public class OptimizedBubbleSort{

public static void bubblesort(int[] arr){

int n=arr.length;

boolean swapped;

for(int i=0;i<n-1;i++){

swapped=false;

for(int j=0;j<n-1;j++){

if(arr[j]>arr[j+1]){

int temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

swapped=true;

}

}

if(!swapped)break;

}

}

public static void main(String[] args){

int[]arr={1,2,3,4,5};

bubblesort(arr);

for(int num:arr){

System.out.print(num+" ");

}

}

}

MIN HEAP:

import java.util.ArrayList;

class MinHeap {

    private ArrayList<Integer> heap;

    public MinHeap() {

        heap = new ArrayList<>();

    }

    // Insert a new value into the heap

    public void insert(int val) {

        heap.add(val);

        int currentIndex = heap.size() - 1;

        // Bubble up to maintain min heap property

        while (currentIndex > 0) {

            int parentIndex = (currentIndex - 1) / 2;

            if (heap.get(parentIndex) > heap.get(currentIndex)) {

                swap(parentIndex, currentIndex);

                currentIndex = parentIndex;

            } else {

                break;

            }

        }

    }

    // Remove and return the min (root) element from the heap

    public int extractMin() {

        if (heap.size() == 0)

            throw new IllegalStateException("Heap is empty");

        int min = heap.get(0);

        // Move last element to root and remove last

        int lastIndex = heap.size() - 1;

        heap.set(0, heap.get(lastIndex));

        heap.remove(lastIndex);

        // Heapify down from root

        heapifyDown(0);

        return min;

    }

    private void heapifyDown(int index) {

        int size = heap.size();

        int smallest = index;

        while (true) {

            int left = 2 \* index + 1;

            int right = 2 \* index + 2;

            if (left < size && heap.get(left) < heap.get(smallest)) {

                smallest = left;

            }

            if (right < size && heap.get(right) < heap.get(smallest)) {

                smallest = right;

            }

            if (smallest != index) {

                swap(index, smallest);

                index = smallest;

            } else {

                break;

            }

        }

    }

    private void swap(int i, int j) {

        int temp = heap.get(i);

        heap.set(i, heap.get(j));

        heap.set(j, temp);

    }

    public boolean isEmpty() {

        return heap.size() == 0;

    }

    public void printHeap() {

        System.out.println(heap);

    }

    public static void main(String[] args) {

        MinHeap minHeap = new MinHeap();

        // Insert values

        minHeap.insert(10);

        minHeap.insert(4);

        minHeap.insert(15);

        minHeap.insert(20);

        minHeap.insert(0);

        minHeap.insert(8);

        System.out.println("MinHeap elements after insertions:");

        minHeap.printHeap();

        System.out.println("\nExtracting min repeatedly:");

        while (!minHeap.isEmpty()) {

            int min = minHeap.extractMin();

            System.out.println("Extracted Min: " + min);

            System.out.print("Heap now: ");

            minHeap.printHeap();

        }

    }

}

TREE NODE:

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int val) {

        this.val = val;

        left = right = null;

    }

}

public class Tree {

    // Method to search for a value in the binary tree

    public boolean search(TreeNode root, int key) {

        if (root == null) {

            return false; // Tree is empty or reached leaf node

        }

        if (root.val == key) {

            return true; // Key found

        }

        // Recursively search in left and right subtrees

        return search(root.left, key) || search(root.right, key);

    }

    public static void main(String[] args) {

        BinaryTree tree = new BinaryTree();

        // Constructing example binary tree:

        //       10

        //      /  \

        //     20   5

        //    /    /  \

        //   30   15   7

        TreeNode root = new TreeNode(10);

        root.left = new TreeNode(20);

        root.right = new TreeNode(5);

        root.left.left = new TreeNode(30);

        root.right.left = new TreeNode(15);

        root.right.right = new TreeNode(7);

        int searchKey = 15;

        boolean result = tree.search(root, searchKey);

        System.out.println("Is " + searchKey + " found in the tree? " + result);

    }

}

SCANNERREADEXAMPLE:

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class ScannerReadExample {

    public static void main(String[] args) {

        Scanner inputScanner = new Scanner(System.in);

        // Reading input from the user

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

        String name = inputScanner.nextLine();

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

        int age = inputScanner.nextInt();

        System.out.println("Hello " + name + ", you are " + age + " years old.");

        inputScanner.close();

        // Reading from a file using Scanner

        try {

            File file = new File("sample.txt"); // Ensure this file exists

            Scanner fileScanner = new Scanner(file);

            System.out.println("\nContents of sample.txt:");

            while (fileScanner.hasNextLine()) {

                String line = fileScanner.nextLine();

                System.out.println(line);

            }

            fileScanner.close();

        } catch (FileNotFoundException e) {

            System.out.println("File not found. Please make sure 'sample.txt' exists.");

        }

    }

HEAPCHECKER:

public class HeapChecker {

    // Check if array arr represents a max-heap

    public static boolean isMaxHeap(int[] arr) {

        int n = arr.length;

        // Start from root and go till the last parent node

        for (int i = 0; i <= (n - 2) / 2; i++) {

            int left = 2 \* i + 1;

            int right = 2 \* i + 2;

            // If left child exists and is greater than parent

            if (left < n && arr[i] < arr[left]) {

                return false;

            }

            // If right child exists and is greater than parent

            if (right < n && arr[i] < arr[right]) {

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        int[] heapArray = {90, 15, 10, 7, 12, 2};

        System.out.println("Is max heap? " + isMaxHeap(heapArray));

    }

}

INSERTIONSORT:

public class InsertionSort{

public static void insertionSort(int[] arr) {

for(int i=1;i<arr.length;i++){

int key=arr[i];

int j=i-1;

while(j >=0 && arr[j] > key){

arr[j+1]=arr[j];

j--;

}

arr[j+1]=key;

}

}

public static void main(String[] args){

int[] arr={5,3,1,4,2};

insertionSort(arr);

for(int num:arr){

System.out.print(num+" ");

}

}

}

BINARYTREECOUNTNODES:

class Node {

    int val;

    Node left, right;

    Node(int val) {

        this.val = val;

        left = right = null;

    }

}

public class BinaryTreeCountNodes {

    Node root;

    int countNodes(Node node) {

        if (node == null)

            return 0;

        return 1 + countNodes(node.left) + countNodes(node.right);

    }

    public static void main(String[] args) {

        BinaryTreeCountNodes tree = new BinaryTreeCountNodes();

        tree.root = new Node(1);

        tree.root.left = new Node(2);

        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);

        tree.root.left.right = new Node(5);

        int totalNodes = tree.countNodes(tree.root);

        System.out.println("Total number of nodes in the binary tree: " + totalNodes);

    }

}

BINARYLEAFCOUNT:

class Node {

    int val;

    Node left, right;

    Node(int val) {

        this.val = val;

        left = right = null;

    }

}

public class BinaryTreeLeafCount {

    Node root;

    int countLeafNodes(Node node) {

        if (node == null)

            return 0;

        if (node.left == null && node.right == null)

            return 1;

        return countLeafNodes(node.left) + countLeafNodes(node.right);

    }

    public static void main(String[] args) {

        BinaryTreeLeafCount tree = new BinaryTreeLeafCount();

        tree.root = new Node(1);

        tree.root.left = new Node(2);

        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);

        tree.root.left.right = new Node(5);

        int leafCount = tree.countLeafNodes(tree.root);

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

    }

}

BINARYTREESUM:

class Node {

    int val;

    Node left, right;

    Node(int val) {

        this.val = val;

        left = right = null;

    }

}

public class BinaryTreeSum {

    Node root;

    int sumOfNodes(Node node) {

        if (node == null)

            return 0;

        return node.val + sumOfNodes(node.left) + sumOfNodes(node.right);

    }

    public static void main(String[] args) {

        BinaryTreeSum tree = new BinaryTreeSum();

        tree.root = new Node(1);

        tree.root.left = new Node(2);

        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);

        tree.root.left.right = new Node(5);

        int totalSum = tree.sumOfNodes(tree.root);

        System.out.println("Sum of all nodes in the binary tree: " + totalSum);

    }

}

BINARYTREE TRAVERSAL:

class Node {

    int val;

    Node left, right;

    Node(int val) {

        this.val = val;

        left = right = null;

    }

}

public class BinaryTreeTraversal {

    Node root;

    void inorder(Node node) {

        if (node == null) return;

        inorder(node.left);

        System.out.print(node.val + " ");

        inorder(node.right);

    }

    void postorder(Node node) {

        if (node == null) return;

        postorder(node.left);

        postorder(node.right);

        System.out.print(node.val + " ");

    }

    public static void main(String[] args) {

        BinaryTreeTraversal tree = new BinaryTreeTraversal();

        tree.root = new Node(1);

        tree.root.left = new Node(2);

        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);

        tree.root.left.right = new Node(5);

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

        tree.inorder(tree.root);

BUFFEREADWRITEEXAMPLE:

import java.io.\*;

public class BufferedReadWriteExample {

    public static void main(String[] args) {

        String fileName = "buffered\_example.txt";

        // Writing to a file using BufferedWriter

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {

            writer.write("This is the first line using BufferedWriter.");

            writer.newLine();  // Adds a new line

            writer.write("This is the second line.");

            System.out.println("File written successfully using BufferedWriter.");

        } catch (IOException e) {

            System.out.println("Error writing to file.");

            e.printStackTrace();

        }

        // Reading from a file using BufferedReader

        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

            System.out.println("\nReading file using BufferedReader:");

            String line;

            while ((line = reader.readLine()) != null) {

                System.out.println(line);

            }

        } catch (IOException e) {

            System.out.println("Error reading the file.");

            e.printStackTrace();

        }

    }

}

COUNTCOMPONENTS:

import java.util.\*;

class Graph {

    private Map<Integer, List<Integer>> adjList;

    // Constructor to initialize the graph

    public Graph() {

        adjList = new HashMap<>();

    }

    // Method to add an edge to the graph

    public void addEdge(int u, int v) {

        adjList.putIfAbsent(u, new ArrayList<>());

        adjList.putIfAbsent(v, new ArrayList<>());

        adjList.get(u).add(v);

        adjList.get(v).add(u); // For undirected graph

    }

    // Method to count the number of connected components

    public int countConnectedComponents() {

        Set<Integer> visited = new HashSet<>();

        int componentCount = 0;

        // Iterate over all nodes in the graph

        for (Integer node : adjList.keySet()) {

            if (!visited.contains(node)) {

                // Start DFS from the unvisited node

                dfs(node, visited);

                componentCount++;  // Increment count for each connected component

            }

        }

        return componentCount;

    }

    // DFS method to visit all nodes in a component

    private void dfs(int node, Set<Integer> visited) {

        visited.add(node);  // Mark the current node as visited

        // Visit all neighbors

        for (int neighbor : adjList.getOrDefault(node, new ArrayList<>())) {

            if (!visited.contains(neighbor)) {

                dfs(neighbor, visited);

            }

        }

    }

}

public class CountComponents {

    public static void main(String[] args) {

        Graph graph = new Graph();

        // Adding edges to the graph

        graph.addEdge(0, 1);

        graph.addEdge(0, 2);

        graph.addEdge(1, 2);

        graph.addEdge(3, 4);

        // Counting the connected components in the graph

        int numComponents = graph.countConnectedComponents();

        System.out.println("Number of connected components: " + numComponents);

    }

}

CYCLEDETECTION:

import java.util.\*;

class Graph {

    private Map<Integer, List<Integer>> adjList;

    // Constructor to initialize the graph

    public Graph() {

        adjList = new HashMap<>();

    }

    // Method to add an edge to the graph

    public void addEdge(int u, int v) {

        adjList.putIfAbsent(u, new ArrayList<>());

        adjList.putIfAbsent(v, new ArrayList<>());

        adjList.get(u).add(v);

        adjList.get(v).add(u);  // For undirected graph

    }

    // Method to check if the graph has a cycle

    public boolean hasCycle() {

        Set<Integer> visited = new HashSet<>();

        for (Integer node : adjList.keySet()) {

            if (!visited.contains(node)) {

                if (dfs(node, visited, -1)) {

                    return true; // Cycle detected

                }

            }

        }

        return false; // No cycle detected

    }

    // DFS method to detect cycle

    private boolean dfs(int current, Set<Integer> visited, int parent) {

        visited.add(current);

        // Explore all the neighbors

        for (int neighbor : adjList.get(current)) {

            if (!visited.contains(neighbor)) {

                // Recursively call DFS for the neighbor

                if (dfs(neighbor, visited, current)) {

                    return true;

                }

            } else if (neighbor != parent) {

                // If the neighbor is visited and is not the parent, cycle is detected

                return true;

            }

        }

        return false;

    }

}

public class CycleDetection {

    public static void main(String[] args) {

        Graph graph = new Graph();

        // Adding edges to the graph

        graph.addEdge(0, 1);

        graph.addEdge(0, 2);

        graph.addEdge(1, 3);

        graph.addEdge(2, 3);

        // Checking for a cycle in the graph

        if (graph.hasCycle()) {

            System.out.println("Cycle detected in the graph.");

        } else {

            System.out.println("No cycle detected in the graph.");

        }

    }

}

FILECLASS:

import java.io.File;

public class Fileclass {

    public static void main(String[] args) {

        File file = new File("test.txt");

        if (file.exists()) {

            System.out.println("File exists.");

        } else {

            System.out.println("File does not exist.");

        }

        System.out.println("File Name: " + file.getName());

        System.out.println("File Path: " + file.getAbsolutePath());

    }

}

INORDER AND POSTORDER:

class Node {

    int val;

    Node left, right;

    Node(int val) {

        this.val = val;

        left = right = null;

    }

}

public class BinaryTreeTraversal {

    Node root;

    void inorder(Node node) {

        if (node == null) return;

        inorder(node.left);

        System.out.print(node.val + " ");

        inorder(node.right);

    }

    void postorder(Node node) {

        if (node == null) return;

        postorder(node.left);

        postorder(node.right);

        System.out.print(node.val + " ");

    }

    public static void main(String[] args) {

        BinaryTreeTraversal tree = new BinaryTreeTraversal();

        tree.root = new Node(1);

        tree.root.left = new Node(2);

        tree.root.right = new Node(3);

        tree.root.left.left = new Node(4);

        tree.root.left.right = new Node(5);

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

        tree.inorder(tree.root);