**Assignment-10**

**M.Mahathi**

**192324098**

1. Create a generic method sort List that takes a list of comparable elements and sorts it. Demonstrate this method with a list of Strings and a list of Integers.

Java code:

import java.util.List;

import java.util.ArrayList;

import java.util.Collections;

public class GenericSort {

public static <T extends Comparable<T>> void sortList(List<T> list) {

Collections.sort(list);

}

public static void main(String[] args) {

List<String> stringList = new ArrayList<>();

stringList.add("Banana");

stringList.add("Apple");

stringList.add("Cherry");

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

sortList(stringList);

System.out.println("After sorting: " + stringList);

List<Integer> intList = new ArrayList<>();

intList.add(4);

intList.add(6);

intList.add(9);

intList.add(2);

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

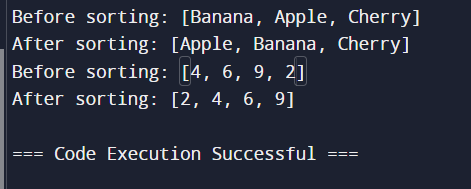
sortList(intList);

System.out.println("After sorting: " + intList);

}

}

OUTPUT:



2. Write agenericclassTreeNoderepresentinganode ina tree withchildren. Implementmethodstoaddchildren, traversethetree (e.g.,depth-firstsearch),andfindanodebyvalue.Demonstratethis withatreeofStringsandIntegers.

Java code:

import java.util.ArrayList;

import java.util.List;

public class TreeNode {

private Integer value;

private List<TreeNode> children;

// Constructor to initialize the node with a value

public TreeNode(Integer value) {

this.value = value;

this.children = new ArrayList<>();

}

// Method to add a child node

public void addChild(TreeNode child) {

children.add(child);

}

// Method to get the value of the node

public Integer getValue() {

return value;

}

// Method to traverse the tree (DFS)

public void traverseDFS() {

System.out.println(value);

for (TreeNode child : children) {

child.traverseDFS();

}

}

// Method to find a node by value (DFS)

public TreeNode findNodeByValue(Integer value) {

if (this.value.equals(value)) {

return this;

}

for (TreeNode child : children) {

TreeNode result = child.findNodeByValue(value);

if (result != null) {

return result;

}

}

return null;

}

public static void main(String[] args) {

// Example usage with a tree of Integers

TreeNode root = new TreeNode(1);

TreeNode child1 = new TreeNode(2);

TreeNode child2 = new TreeNode(3);

root.addChild(child1);

root.addChild(child2);

System.out.println("Traversing tree of Integers:");

root.traverseDFS();

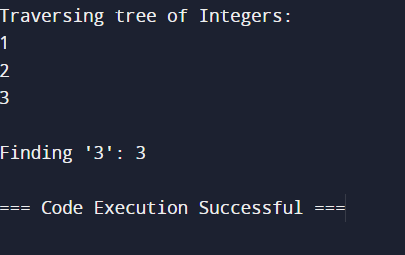
// Using the getValue() method

System.out.println("\nFinding '3': " + root.findNodeByValue(3).getValue());

}

}

OUTPUT:



3. Implement a generic class GenericPriorityQueue>withmethods likeenqueue, dequeue, andpeek. The elements should be dequeued in priorityorder.Demonstrate withIntegerandString.

Java code:

import java.util.PriorityQueue;

public class GenericPriorityQueue<T extends Comparable<T>> {

private PriorityQueue<T> queue;

public GenericPriorityQueue() {

this.queue = new PriorityQueue<>();

}

public void enqueue(T element) {

queue.add(element);

}

public T dequeue() {

return queue.poll();

}

public T peek() {

return queue.peek();

}

public static void main(String[] args) {

GenericPriorityQueue<Integer> intQueue = new GenericPriorityQueue<>();

intQueue.enqueue(34);

intQueue.enqueue(7);

intQueue.enqueue(41);

System.out.println("Integer Queue - Peek: " + intQueue.peek());

System.out.println("Integer Queue - Dequeue: " + intQueue.dequeue());

System.out.println("Integer Queue - Dequeue: " + intQueue.dequeue());

GenericPriorityQueue<String> stringQueue = new GenericPriorityQueue<>();

stringQueue.enqueue("Banana");

stringQueue.enqueue("Apple");

stringQueue.enqueue("Cherry");

System.out.println("\nString Queue - Peek: " + stringQueue.peek());

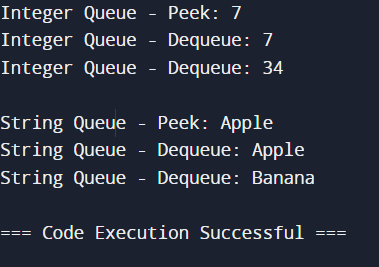
System.out.println("String Queue - Dequeue: " + stringQueue.dequeue());

System.out.println("String Queue - Dequeue: " + stringQueue.dequeue());

}

}

OUTPUT:



4. DesignagenericclassGraphwithmethods foraddingnodes, addingedges,andperforminggraphtraversals(e.g.,BFSandDFS). Ensure that the graph can handle both directed and undirected graphs.DemonstratewithagraphofStringnodesandanothergraph ofIntegernodes.

Java code:

import java.util.\*;

public class Graph<T> {

private Map<T, List<T>> adjacencyList;

private boolean isDirected;

// Constructor to initialize the graph (directed or undirected)

public Graph(boolean isDirected) {

this.adjacencyList = new HashMap<>();

this.isDirected = isDirected;

}

// Method to add a node to the graph

public void addNode(T node) {

adjacencyList.putIfAbsent(node, new ArrayList<>());

}

// Method to add an edge between two nodes

public void addEdge(T source, T destination) {

adjacencyList.get(source).add(destination);

if (!isDirected) {

adjacencyList.get(destination).add(source);

}

}

// Method to perform Depth-First Search (DFS) traversal

public void dfs(T start) {

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

dfsHelper(start, visited);

}

private void dfsHelper(T node, Set<T> visited) {

visited.add(node);

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

for (T neighbor : adjacencyList.get(node)) {

if (!visited.contains(neighbor)) {

dfsHelper(neighbor, visited);

}

}

}

// Method to perform Breadth-First Search (BFS) traversal

public void bfs(T start) {

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

Queue<T> queue = new LinkedList<>();

queue.add(start);

visited.add(start);

while (!queue.isEmpty()) {

T node = queue.poll();

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

for (T neighbor : adjacencyList.get(node)) {

if (!visited.contains(neighbor)) {

visited.add(neighbor);

queue.add(neighbor);

}

}

}

}

public static void main(String[] args) {

// New example with a different String graph

Graph<String> stringGraph = new Graph<>(true); // directed graph

stringGraph.addNode("X");

stringGraph.addNode("Y");

stringGraph.addNode("Z");

stringGraph.addNode("W");

stringGraph.addEdge("X", "Y");

stringGraph.addEdge("X", "Z");

stringGraph.addEdge("Y", "W");

stringGraph.addEdge("Z", "W");

System.out.println("DFS Traversal (String Graph):");

stringGraph.dfs("X"); // Output: X Y W Z (or X Z W Y depending on edge order)

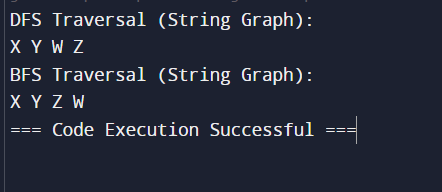
System.out.println("\nBFS Traversal (String Graph):");

stringGraph.bfs("X"); // Output: X Y Z W

}

}

OUTPUT:



5. CreateagenericclassMatrixthat representsa matrix and supports operations like addition, subtraction, and multiplicationofmatrices.Ensure that theoperationsaretype-safe andefficient.DemonstratewithmatricesofIntegerandDouble.

Java code:

public class Matrix<T extends Number> {

private T[][] data;

private int rows;

private int columns;

public Matrix(int rows, int columns) {

this.rows = rows;

this.columns = columns;

this.data = (T[][]) new Number[rows][columns];

}

public void set(int row, int column, T value) {

data[row][column] = value;

}

public T get(int row, int column) {

return data[row][column];

}

public Matrix<T> add(Matrix<T> other) {

checkDimensions(other);

Matrix<T> result = new Matrix<>(rows, columns);

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

result.set(i, j, addValues(this.get(i, j), other.get(i, j)));

}

}

return result;

}

public Matrix<T> subtract(Matrix<T> other) {

checkDimensions(other);

Matrix<T> result = new Matrix<>(rows, columns);

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

result.set(i, j, subtractValues(this.get(i, j), other.get(i, j)));

}

}

return result;

}

public Matrix<Double> multiply(Matrix<T> other) {

if (this.columns != other.rows) {

throw new IllegalArgumentException("Matrix dimensions do not match for multiplication.");

}

Matrix<Double> result = new Matrix<>(this.rows, other.columns);

for (int i = 0; i < this.rows; i++) {

for (int j = 0; j < other.columns; j++) {

double sum = 0;

for (int k = 0; k < this.columns; k++) {

sum += this.get(i, k).doubleValue() \* other.get(k, j).doubleValue();

}

result.set(i, j, sum);

}

}

return result;

}

private T addValues(T a, T b) {

if (a instanceof Integer) return (T) Integer.valueOf(a.intValue() + b.intValue());

if (a instanceof Double) return (T) Double.valueOf(a.doubleValue() + b.doubleValue());

throw new UnsupportedOperationException("Unsupported type for addition.");

}

private T subtractValues(T a, T b) {

if (a instanceof Integer) return (T) Integer.valueOf(a.intValue() - b.intValue());

if (a instanceof Double) return (T) Double.valueOf(a.doubleValue() - b.doubleValue());

throw new UnsupportedOperationException("Unsupported type for subtraction.");

}

private void checkDimensions(Matrix<T> other) {

if (this.rows != other.rows || this.columns != other.columns) {

throw new IllegalArgumentException("Matrix dimensions must match.");

}

}

public void print() {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

System.out.print(get(i, j) + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

Matrix<Integer> intMatrix1 = new Matrix<>(2, 2);

Matrix<Integer> intMatrix2 = new Matrix<>(2, 2);

intMatrix1.set(0, 0, 1); intMatrix1.set(0, 1, 2);

intMatrix1.set(1, 0, 3); intMatrix1.set(1, 1, 4);

intMatrix2.set(0, 0, 5); intMatrix2.set(0, 1, 6);

intMatrix2.set(1, 0, 7); intMatrix2.set(1, 1, 8);

System.out.println("Integer Matrix 1:");

intMatrix1.print();

System.out.println("Integer Matrix 2:");

intMatrix2.print();

Matrix<Integer> intSum = intMatrix1.add(intMatrix2);

System.out.println("Sum:");

intSum.print();

Matrix<Integer> intDiff = intMatrix1.subtract(intMatrix2);

System.out.println("Difference:");

intDiff.print();

Matrix<Double> intProduct = intMatrix1.multiply(intMatrix2);

System.out.println("Product:");

intProduct.print();

Matrix<Double> doubleMatrix1 = new Matrix<>(2, 2);

Matrix<Double> doubleMatrix2 = new Matrix<>(2, 2);

doubleMatrix1.set(0, 0, 1.5); doubleMatrix1.set(0, 1, 2.5);

doubleMatrix1.set(1, 0, 3.5); doubleMatrix1.set(1, 1, 4.5);

doubleMatrix2.set(0, 0, 5.5); doubleMatrix2.set(0, 1, 6.5);

doubleMatrix2.set(1, 0, 7.5); doubleMatrix2.set(1, 1, 8.5);

System.out.println("\nDouble Matrix 1:");

doubleMatrix1.print();

System.out.println("Double Matrix 2:");

doubleMatrix2.print();

Matrix<Double> doubleSum = doubleMatrix1.add(doubleMatrix2);

System.out.println("Sum:");

doubleSum.print();

Matrix<Double> doubleDiff = doubleMatrix1.subtract(doubleMatrix2);

System.out.println("Difference:");

doubleDiff.print();

Matrix<Double> doubleProduct = doubleMatrix1.multiply(doubleMatrix2);

System.out.println("Product:");

doubleProduct.print();

}

}

OUTPUT:

