ASSIGNMENT-10:
v.sree ruthin reddy
192324112

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

Code:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class GenericSorter {

    // Generic method to sort a list of comparable elements
    public static <T extends Comparable<T>> void sortList(List<T> list) {
        Collections.sort(list);
    }

    public static void main(String[] args) {
        // Demonstration with a list of Strings
        List<String> stringList = new ArrayList<>();
        stringList.add("Banana");
        stringList.add("Apple");
        stringList.add("Orange");
        stringList.add("Mango");

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

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

        // Demonstration with a list of Integers
        List<Integer> integerList = new ArrayList<>();
        integerList.add(42);
        integerList.add(5);
        integerList.add(16);
        integerList.add(8);

        System.out.println("\nBefore sorting (Integers): " + integerList);
        sortList(integerList);
        System.out.println("After sorting (Integers): " + integerList);
    }
}
```

Output:

Programiz
Online Java Compiler

Main.java | Output

```
java -cp /tmp/KOX8UhFjza/GenericSorter
Before sorting (Strings): [Banana, Apple, Orange, Mango]
After sorting (Strings): [Apple, Banana, Mango, Orange]

Before sorting (Integers): [42, 5, 16, 8]
After sorting (Integers): [5, 8, 16, 42]

=== Code Execution Successful ===
```

Generic Method Declaration:

- <T extends Comparable<T>> specifies that the method accepts any type T that implements the Comparable<T> interface. This ensures that the elements in the list can be compared to each other.

- The method sortList(List<T> list) takes a list of elements of type T and sorts it using Collections.sort.

Demonstration:

- A list of Strings (stringList) is created and populated with some values. The sortList method is called to sort this list.
- Similarly, a list of Integers (integerList) is created, populated, and sorted using the same sortList method.

2. Write a generic class TreeNode<T> representing a node in a tree with children. Implement methods to add children, traverse the tree (e.g., depth-first search), and find a node by value. Demonstrate this with a tree of Strings and Integers.

Code:

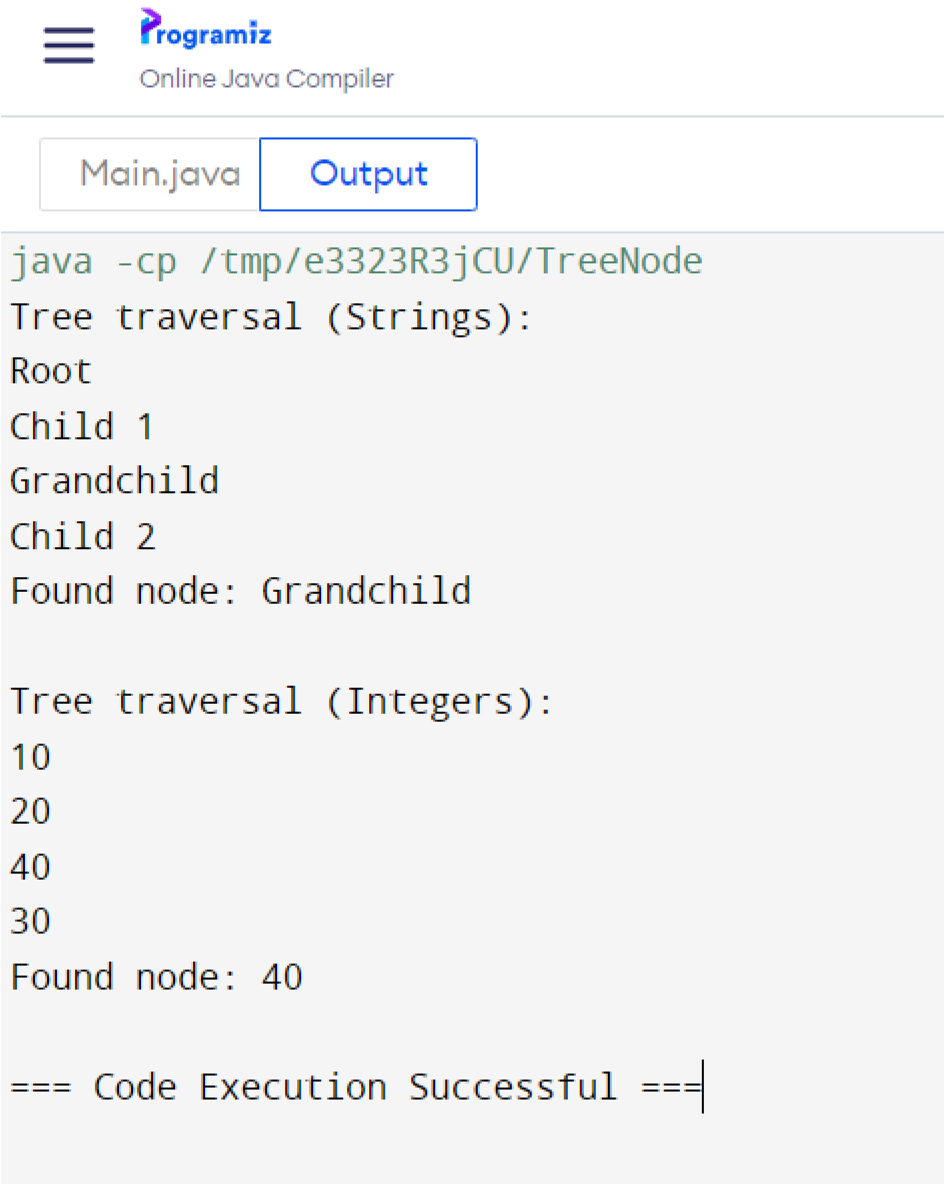
```
import java.util.ArrayList;

import java.util.List;

import java.util.Optional;


public class TreeNode<T> {

    private T value;

    private List<TreeNode<T>> children;


    // Constructor to initialize the node with a value

    public TreeNode(T value) {

        this.value = value;

        this.children = new ArrayList<>();

    }


    // Getter for the node's value

    public T getValue() {

        return value;
```

```
}

// Getter for the list of children
public List<TreeNode<T>> getChildren() {
    return children;
}

// Method to add a child node
public void addChild(TreeNode<T> child) {
    children.add(child);
}

// Method for depth-first search (DFS) traversal
public void traverse(TreeNode<T> node) {
    System.out.println(node.getValue());
    for (TreeNode<T> child : node.getChildren()) {
        traverse(child);
    }
}

// Method to find a node by value using DFS
public Optional<TreeNode<T>> findNode(TreeNode<T> node, T value) {
    if (node.getValue().equals(value)) {
        return Optional.of(node);
    }
    for (TreeNode<T> child : node.getChildren()) {
        Optional<TreeNode<T>> result = findNode(child, value);
        if (result.isPresent()) {
            return result;
        }
```

```
        }
        return Optional.empty();
    }

    // Main method to demonstrate the TreeNode class
    public static void main(String[] args) {
        // Demonstration with a tree of Strings
        TreeNode<String> rootString = new TreeNode<>("Root");
        TreeNode<String> child1String = new TreeNode<>("Child 1");
        TreeNode<String> child2String = new TreeNode<>("Child 2");
        TreeNode<String> grandChildString = new TreeNode<>("Grandchild");

        rootString.addChild(child1String);
        rootString.addChild(child2String);
        child1String.addChild(grandChildString);

        System.out.println("Tree traversal (Strings):");
        rootString.traverse(rootString);

        Optional<TreeNode<String>> foundNodeString = rootString.findNode(rootString,
"Grandchild");
        System.out.println("Found node: " +
foundNodeString.map(TreeNode::getValue).orElse("Not Found"));

        // Demonstration with a tree of Integers
        TreeNode<Integer> rootInt = new TreeNode<>(10);
        TreeNode<Integer> child1Int = new TreeNode<>(20);
        TreeNode<Integer> child2Int = new TreeNode<>(30);
        TreeNode<Integer> grandChildInt = new TreeNode<>(40);

        rootInt.addChild(child1Int);
```

```
        rootInt.addChild(child2Int);
        child1Int.addChild(grandChildInt);

        System.out.println("\nTree traversal (Integers):");
        rootInt.traverse(rootInt);

        Optional<TreeNode<Integer>> foundNodeInt = rootInt.findNode(rootInt, 40);
        System.out.println("Found node: " + foundNodeInt.map(TreeNode::getValue).orElse(-
1));  // orElse(-1) as a default value
    }
}
```

Output:



3. Implement a generic class GenericPriorityQueue<T extends Comparable<T>> with methods like enqueue, dequeue, and peek. The elements should be dequeued in priority order. Demonstrate with Integer and String.

Code:

```
import java.util.ArrayList;
import java.util.List;

public class GenericPriorityQueue<T extends Comparable<T>> {
    private List<T> heap;
```

```
public GenericPriorityQueue() {
    this.heap = new ArrayList<>();
}

// Method to enqueue an element into the priority queue
public void enqueue(T value) {
    heap.add(value);
    heapifyUp(heap.size() - 1);
}

// Method to dequeue the highest priority element (smallest element)
public T dequeue() {
    if (heap.isEmpty()) {
        throw new IllegalStateException("Queue is empty");
    }

    T root = heap.get(0);
    T lastItem = heap.remove(heap.size() - 1);

    if (!heap.isEmpty()) {
        heap.set(0, lastItem);
        heapifyDown(0);
    }

    return root;
}

// Method to peek at the highest priority element without removing it
public T peek() {
    if (heap.isEmpty()) {
        throw new IllegalStateException("Queue is empty");
```

```java
    }
    return heap.get(0);
}

// Helper method to maintain heap order after enqueuing (bubble up)
private void heapifyUp(int index) {
    T current = heap.get(index);
    int parentIndex = (index - 1) / 2;

    while (index > 0 && heap.get(parentIndex).compareTo(current) > 0) {
        heap.set(index, heap.get(parentIndex));
        index = parentIndex;
        parentIndex = (index - 1) / 2;
    }
    heap.set(index, current);
}

// Helper method to maintain heap order after dequeuing (bubble down)
private void heapifyDown(int index) {
    T current = heap.get(index);
    int size = heap.size();

    while (true) {
        int leftChild = 2 * index + 1;
        int rightChild = 2 * index + 2;
        int smallest = index;

        if (leftChild < size && heap.get(leftChild).compareTo(heap.get(smallest)) < 0) {
            smallest = leftChild;
        }

        if (rightChild < size && heap.get(rightChild).compareTo(heap.get(smallest)) < 0) {
```

```
            smallest = rightChild;
        }

        if (smallest == index) {
            break;
        }

        heap.set(index, heap.get(smallest));
        index = smallest;
    }
    heap.set(index, current);
}

// Method to check if the queue is empty
public boolean isEmpty() {
    return heap.isEmpty();
}

public static void main(String[] args) {
    // Demonstration with Integer
    GenericPriorityQueue<Integer> intQueue = new GenericPriorityQueue<>();
    intQueue.enqueue(5);
    intQueue.enqueue(3);
    intQueue.enqueue(8);
    intQueue.enqueue(1);

    System.out.println("Peek (Integer): " + intQueue.peek());
    while (!intQueue.isEmpty()) {
        System.out.println("Dequeue (Integer): " + intQueue.dequeue());
    }

    // Demonstration with String
```

```java
        GenericPriorityQueue<String> stringQueue = new GenericPriorityQueue<>();
        stringQueue.enqueue("apple");
        stringQueue.enqueue("banana");
        stringQueue.enqueue("cherry");
        stringQueue.enqueue("date");

        System.out.println("\nPeek (String): " + stringQueue.peek());
        while (!stringQueue.isEmpty()) {
            System.out.println("Dequeue (String): " + stringQueue.dequeue());
        }
    }
}
```

Output:

Output

```
java -cp /tmp/2hxu4wX2Dx/GenericPriorityQueue
Peek (Integer): 1
Dequeue (Integer): 1
Dequeue (Integer): 3
Dequeue (Integer): 5
Dequeue (Integer): 8

Peek (String): apple
Dequeue (String): apple
Dequeue (String): banana
Dequeue (String): cherry
Dequeue (String): date

=== Code Execution Successful ===
```

4. Design a generic class Graph<T> with methods for adding nodes, adding edges, and performing graph traversals (e.g., BFS and DFS).

Ensure that the graph can handle both directed and undirected graphs. Demonstrate with a graph of String nodes and another graph of Integer nodes.

Code:

```
import java.util.*;

public class Graph<T> {
    private Map<T, List<T>> adjacencyList;
    private boolean isDirected;

    // Constructor to initialize the graph as 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 from, T to) {
        adjacencyList.get(from).add(to);
        if (!isDirected) {
            adjacencyList.get(to).add(from);
        }
    }

    // Method to perform Breadth-First Search (BFS)
    public void bfs(T startNode) {
```

```java
    Set<T> visited = new HashSet<>();
    Queue<T> queue = new LinkedList<>();
    queue.add(startNode);
    visited.add(startNode);

    while (!queue.isEmpty()) {
        T node = queue.poll();
        System.out.println(node);

        for (T neighbor : adjacencyList.get(node)) {
            if (!visited.contains(neighbor)) {
                visited.add(neighbor);
                queue.add(neighbor);
            }
        }
    }
}

// Method to perform Depth-First Search (DFS)
public void dfs(T startNode) {
    Set<T> visited = new HashSet<>();
    dfsHelper(startNode, visited);
}

// Helper method for DFS using recursion
private void dfsHelper(T node, Set<T> visited) {
    visited.add(node);
    System.out.println(node);

    for (T neighbor : adjacencyList.get(node)) {
        if (!visited.contains(neighbor)) {
            dfsHelper(neighbor, visited);
```

```
            }
        }
    }

    // Main method to demonstrate the Graph class
    public static void main(String[] args) {
        // Demonstration with a graph of Strings (undirected)
        Graph<String> stringGraph = new Graph<>(false);
        stringGraph.addNode("A");
        stringGraph.addNode("B");
        stringGraph.addNode("C");
        stringGraph.addNode("D");

        stringGraph.addEdge("A", "B");
        stringGraph.addEdge("A", "C");
        stringGraph.addEdge("B", "D");
        stringGraph.addEdge("C", "D");

        System.out.println("BFS traversal (Strings):");
        stringGraph.bfs("A");

        System.out.println("\nDFS traversal (Strings):");
        stringGraph.dfs("A");

        // Demonstration with a graph of Integers (directed)
        Graph<Integer> intGraph = new Graph<>(true);
        intGraph.addNode(1);
        intGraph.addNode(2);
        intGraph.addNode(3);
        intGraph.addNode(4);

        intGraph.addEdge(1, 2);
```

```
        intGraph.addEdge(1, 3);
        intGraph.addEdge(2, 4);
        intGraph.addEdge(3, 4);

        System.out.println("\nBFS traversal (Integers):");
        intGraph.bfs(1);

        System.out.println("\nDFS traversal (Integers):");
        intGraph.dfs(1);
    }
}
```

Output:

Programiz

Online Java Compiler

Main.java | Output

```
java -cp /tmp/1kRNVlGTJU/Graph
BFS traversal (Strings):
A
B
C
D

DFS traversal (Strings):
A
B
D
C

BFS traversal (Integers):
1
2
3
4

DFS traversal (Integers):
1
2
4
3

=== Code Execution Successful ===
```

5. Create a generic class Matrix<T extends Number> that represents a
matrix and supports operations like addition, subtraction, and
multiplication of matrices. Ensure that the operations are type-safe
and efficient. Demonstrate with matrices of Integer and Double.

Code:

```
import java.util.Arrays;

public class Matrix<T extends Number> {
    private T[][] data;
    private int rows;
    private int cols;

    public Matrix(T[][] data) {
        this.data = data;
        this.rows = data.length;
        this.cols = data[0].length;
    }

    // Add two matrices
    public Matrix<T> add(Matrix<T> other) {
        if (this.rows != other.rows || this.cols != other.cols) {
            throw new IllegalArgumentException("Matrices must have the same dimensions for
addition.");
        }

        T[][] result = (T[][]) new Number[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = (T) addNumbers(this.data[i][j], other.data[i][j]);
            }
```

```
        }

        return new Matrix<>(result);
    }

    // Subtract two matrices
    public Matrix<T> subtract(Matrix<T> other) {
        if (this.rows != other.rows || this.cols != other.cols) {
            throw new IllegalArgumentException("Matrices must have the same dimensions for
subtraction.");
        }

        T[][] result = (T[][]) new Number[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = (T) subtractNumbers(this.data[i][j], other.data[i][j]);
            }
        }

        return new Matrix<>(result);
    }

    // Multiply two matrices
    public Matrix<T> multiply(Matrix<T> other) {
        if (this.cols != other.rows) {
            throw new IllegalArgumentException("Matrices must have compatible dimensions for
multiplication.");
        }

        T[][] result = (T[][]) new Number[this.rows][other.cols];

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

```java
        for (int j = 0; j < other.cols; j++) {
            result[i][j] = (T) multiplyAndSumRows(this.data[i], getColumn(other.data, j));
        }
    }

    return new Matrix<>(result);
}

// Helper methods for basic arithmetic operations
private Number addNumbers(Number a, Number b) {
    if (a instanceof Integer) {
        return a.intValue() + b.intValue();
    } else if (a instanceof Double) {
        return a.doubleValue() + b.doubleValue();
    } else {
        throw new UnsupportedOperationException("Type not supported for addition");
    }
}

private Number subtractNumbers(Number a, Number b) {
    if (a instanceof Integer) {
        return a.intValue() - b.intValue();
    } else if (a instanceof Double) {
        return a.doubleValue() - b.doubleValue();
    } else {
        throw new UnsupportedOperationException("Type not supported for subtraction");
    }
}

private Number multiplyAndSumRows(Number[] row, Number[] column) {
    Number sum = 0;
```

```
    for (int i = 0; i < row.length; i++) {
        sum = sum.doubleValue() + row[i].doubleValue() * column[i].doubleValue();
    }

    return sum;
}

private Number[] getColumn(T[][] matrix, int col) {
    Number[] column = new Number[matrix.length];

    for (int i = 0; i < matrix.length; i++) {
        column[i] = matrix[i][col];
    }

    return column;
}

// Method to print the matrix
public void printMatrix() {
    for (T[] row : data) {
        System.out.println(Arrays.toString(row));
    }
}

public static void main(String[] args) {
    Integer[][] intData1 = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
    Integer[][] intData2 = { {9, 8, 7}, {6, 5, 4}, {3, 2, 1} };

    Matrix<Integer> intMatrix1 = new Matrix<>(intData1);
    Matrix<Integer> intMatrix2 = new Matrix<>(intData2);

    System.out.println("Integer Matrix Addition:");
```

```java
        Matrix<Integer> intAddResult = intMatrix1.add(intMatrix2);
        intAddResult.printMatrix();

        System.out.println("\nInteger Matrix Subtraction:");
        Matrix<Integer> intSubtractResult = intMatrix1.subtract(intMatrix2);
        intSubtractResult.printMatrix();

        System.out.println("\nInteger Matrix Multiplication:");
        Matrix<Integer> intMultiplyResult = intMatrix1.multiply(intMatrix2);
        intMultiplyResult.printMatrix();

        Double[][] doubleData1 = { {1.1, 2.2, 3.3}, {4.4, 5.5, 6.6}, {7.7, 8.8, 9.9} };
        Double[][] doubleData2 = { {9.9, 8.8, 7.7}, {6.6, 5.5, 4.4}, {3.3, 2.2, 1.1} };

        Matrix<Double> doubleMatrix1 = new Matrix<>(doubleData1);
        Matrix<Double> doubleMatrix2 = new Matrix<>(doubleData2);

        System.out.println("\nDouble Matrix Addition:");
        Matrix<Double> doubleAddResult = doubleMatrix1.add(doubleMatrix2);
        doubleAddResult.printMatrix();

        System.out.println("\nDouble Matrix Subtraction:");
        Matrix<Double> doubleSubtractResult = doubleMatrix1.subtract(doubleMatrix2);
        doubleSubtractResult.printMatrix();

        System.out.println("\nDouble Matrix Multiplication:");
        Matrix<Double> doubleMultiplyResult = doubleMatrix1.multiply(doubleMatrix2);
        doubleMultiplyResult.printMatrix();
    }
}
```

Output:

Output

```
Note: /tmp/2yqhAeuWgr/Matrix.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
java -cp /tmp/2yqhAeuWgr/Matrix
Integer Matrix Addition:
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]

Integer Matrix Subtraction:
[-8, -6, -4]
[-2, 0, 2]
[4, 6, 8]

Integer Matrix Multiplication:
[30.0, 24.0, 18.0]
[84.0, 69.0, 54.0]
[138.0, 114.0, 90.0]

Double Matrix Addition:
[11.0, 11.0, 11.0]
[11.0, 11.0, 11.0]
[11.0, 11.0, 11.0]

Double Matrix Subtraction:
[-8.8, -6.6000000000000005, -4.4]
[-2.1999999999999993, 0.0, 2.1999999999999993]
[4.4, 6.6000000000000005, 8.8]
```