CSA0961 Programming in java for Distributed Applications

A.MANOJ REDDY 192311171

ASSIGNMENT-10

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.

Program:

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

public class SortListExample {

    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");

        List<Integer> integerList = new ArrayList<>();
        integerList.add(5);
        integerList.add(1);
```

```
        integerList.add(3);

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

        sortList(stringList);
        sortList(integerList);

        System.out.println("After sorting:");
        System.out.println(stringList);
        System.out.println(integerList);
    }
}
```

Output:

```
java -cp /tmp/nwW6TNmBs7/SortListExample
Before sorting:
[Banana, Apple, Cherry]
[5, 1, 3]
After sorting:
[Apple, Banana, Cherry]
[1, 3, 5]

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

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.

Program:

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

class TreeNode<T> {
    T value;
    List<TreeNode<T>> children;

    public TreeNode(T value) {
        this.value = value;
        this.children = new ArrayList<>();
```

```
    }

    public void addChild(TreeNode<T> child) {
        children.add(child);
    }

    public void depthFirstTraversal() {
        System.out.print(value + " ");
        for (TreeNode<T> child : children) {
            child.depthFirstTraversal();
        }
    }

    public TreeNode<T> findNode(T value) {
        if (this.value.equals(value)) {
            return this;
        }
        for (TreeNode<T> child : children) {
            TreeNode<T> result = child.findNode(value);
            if (result != null) {
                return result;
            }
        }
        return null;
    }

    public static void main(String[] args) {
        TreeNode<String> root = new TreeNode<>("Root");
        TreeNode<String> child1 = new TreeNode<>("Child1");
        TreeNode<String> child2 = new TreeNode<>("Child2");
        root.addChild(child1);
        root.addChild(child2);
        child1.addChild(new TreeNode<>("Child1.1"));
        child2.addChild(new TreeNode<>("Child2.1"));

        System.out.println("Depth-First Traversal:");
        root.depthFirstTraversal();

        System.out.println("\nFind Node:");
        TreeNode<String> foundNode = root.findNode("Child2.1");
        if (foundNode != null) {
            System.out.println("Node found: " + foundNode.value);
        } else {
            System.out.println("Node not found");
        }
    }
}
```

Output:

```
java -cp /tmp/Q2hYyIQYbI/TreeNode
Depth-First Traversal:
Root Child1 Child1.1 Child2 Child2.1
Find Node:
Node found: Child2.1

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

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.

Program:

```
import java.util.PriorityQueue;

class GenericPriorityQueue<T extends Comparable<T>> {
   private PriorityQueue<T> queue;

   public GenericPriorityQueue() {
      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(5);
      intQueue.enqueue(1);
      intQueue.enqueue(3);

      System.out.println("Integer Queue:");
      while (intQueue.peek() != null) {
         System.out.println(intQueue.dequeue());
      }
```

```
        GenericPriorityQueue<String> stringQueue = new GenericPriorityQueue<>();
        stringQueue.enqueue("Banana");
        stringQueue.enqueue("Apple");
        stringQueue.enqueue("Cherry");

        System.out.println("String Queue:");
        while (stringQueue.peek() != null) {
            System.out.println(stringQueue.dequeue());
        }
    }
}
```

Output:

```
java -cp /tmp/rK1m4qNKoO/GenericPriorityQueue
Integer Queue:
1
3
5
String Queue:
Apple
Banana
Cherry

=== 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

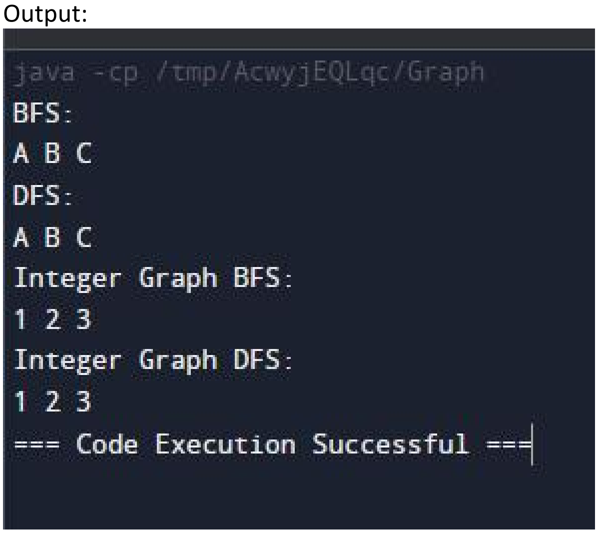
of Integer nodes.

Program:

```
import java.util.*;

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

    public Graph(boolean isDirected) {
        this.adjacencyList = new HashMap<>();
        this.isDirected = isDirected;
    }

    public void addNode(T node) {
        adjacencyList.putIfAbsent(node, new ArrayList<>());
    }
```

```java
public void addEdge(T from, T to) {
    adjacencyList.get(from).add(to);
    if (!isDirected) {
        adjacencyList.get(to).add(from);
    }
}

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 void dfs(T start) {
    Set<T> visited = new HashSet<>();
    Stack<T> stack = new Stack<>();
    stack.push(start);

    while (!stack.isEmpty()) {
        T node = stack.pop();
        if (!visited.contains(node)) {
            visited.add(node);
            System.out.print(node + " ");
            for (T neighbor : adjacencyList.get(node)) {
                if (!visited.contains(neighbor)) {
                    stack.push(neighbor);
                }
            }
        }
    }
}

public static void main(String[] args) {
    Graph<String> stringGraph = new Graph<>(false);
    stringGraph.addNode("A");
    stringGraph.addNode("B");
    stringGraph.addNode("C");
    stringGraph.addEdge("A", "B");
    stringGraph.addEdge("B", "C");

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

    System.out.println("\nDFS:");
```

```
        stringGraph.dfs("A");

        Graph<Integer> intGraph = new Graph<>(true);
        intGraph.addNode(1);
        intGraph.addNode(2);
        intGraph.addNode(3);
        intGraph.addEdge(1, 2);
        intGraph.addEdge(2, 3);

        System.out.println("\nInteger Graph BFS:");
        intGraph.bfs(1);

        System.out.println("\nInteger Graph DFS:");
        intGraph.dfs(1);
    }
}
```
Output:



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.

Program:
```
class Matrix<T extends Number> {
    private T[][] data;
    private int rows, cols;

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

    public Matrix<T> add(Matrix<T> other) {
        T[][] result = (T[][]) new Number[rows][cols];
```

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

public Matrix<T> subtract(Matrix<T> other) {
    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(data[i][j], other.data[i][j]);
        }
    }
    return new Matrix<>(result);
}

public Matrix<T> multiply(Matrix<T> other) {
    T[][] result = (T[][]) new Number[rows][other.cols];
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < other.cols; j++) {
            result[i][j] = (T) new Integer(0);
            for (int k = 0; k < cols; k++) {
                result[i][j] = (T) addNumbers(result[i][j], multiplyNumbers(data[i][k], other.data[k][j]));
            }
        }
    }
    return new Matrix<>(result);
}

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();
    }
    throw new UnsupportedOperationException("Unsupported number type");
}

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();
    }
    throw new UnsupportedOperationException("Unsupported number type");
}

private Number multiplyNumbers(Number a, Number b) {
    if (a instanceof Integer) {
        return a.intValue() * b.intValue();
    } else if (a instanceof Double) {
        return a.doubleValue() * b.doubleValue();
    }
    throw new UnsupportedOperationException("Unsupported number type");
}
```

```
    public void print() {
        for (T[] row : data) {
            for (T val : row) {
                System.out.print(val + " ");
            }
            System.out.println();
        }
    }

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

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

        System.out.println("Integer Matrix Addition:");
        Matrix<Integer> intAddResult = intMatrix1.add(intMatrix2);
        intAddResult.print();

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

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

        Double[][] doubleData1 = {{1.5, 2.5}, {3.5, 4.5}};
        Double[][] doubleData2 = {{5.5, 6.5}, {7.5, 8.5}};

        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.print();

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

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

Output:

```
Note: /tmp/8FenN3dutb/Matrix.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: /tmp/8FenN3dutb/Matrix.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
java -cp /tmp/8FenN3dutb/Matrix
Integer Matrix Addition:
6 8
10 12

Integer Matrix Subtraction:
-4 -4
-4 -4

Integer Matrix Multiplication:
19 22
43 50

Double Matrix Addition:
7.0 9.0
11.0 13.0

Double Matrix Subtraction:
-4.0 -4.0
-4.0 -4.0

Double Matrix Multiplication:
26 30
52 60

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