Assignment-6 Data Structure and Integration in Program

Subject: CSW2 (CSE 2141)

Name: Arpit Kumar Mohanty

Registration Number: 2341013237

Section: 23412G1

Branch: CSE

Q1. Design a class BSTNode in Java with a member variable info to store an integer and two references, left and right, to represent its left and right children. Implement a constructor to initialize these attributes. Develop a method to insert a node while maintaining the properties of a binary search tree. Extend this implementation by adding methods for traversing the tree using pre-order, in-order, and post-order techniques. Finally, add a main method to create a binary search tree, insert multiple nodes, and invoke the traversal methods to display the tree structure.

Solution:

P1_BSTNode.java | P2_CreateTree.java | P3.java | P4_BSTCountry.java | Q5_RemoveNode_BST.java

P1_BSTNode.java > P1_BSTNode > main(String[])

```java
class BSTNode {
    int info;
    BSTNode left, right;

    public BSTNode(int info) {
        this.info = info;
        left = right = null;
    }
}

public class P1_BSTNode {
    private BSTNode root;

    public void insert(int value) {
        root = insertRec(root, value);
    }

    private BSTNode insertRec(BSTNode node, int value) {
        if (node == null) return new BSTNode(value);
        if (value < node.info) {
            node.left = insertRec(node.left, value);
        }
        else if (value > node.info) {  // Prevent duplicates
            node.right = insertRec(node.right, value);
        }
        return node;
    }

    public void preOrder() {
        preOrderRec(root);
        System.out.println();
    }

    private void preOrderRec(BSTNode node) {
        if (node == null) return;
        System.out.print(node.info + " ");
        preOrderRec(node.left);
        preOrderRec(node.right);
    }

    public void inOrder() {
        inOrderRec(root);
        System.out.println();
    }
```

P1_BSTNode.java | P2_CreateTree.java | P3.java | P4_BSTCountry.java | Q5_RemoveNode_BST.java

P1_BSTNode.java > P1_BSTNode > main(String[])

```java
public class P1_BSTNode {
    private void inOrderRec(BSTNode node) {
        if (node == null) return;
        inOrderRec(node.left);
        System.out.print(node.info + " ");
        inOrderRec(node.right);
    }

    public void postOrder() {
        postOrderRec(root);
        System.out.println();
    }

    private void postOrderRec(BSTNode node) {
        if (node == null) return;
        postOrderRec(node.left);
        postOrderRec(node.right);
        System.out.print(node.info + " ");
    }

    Run | Debug
    public static void main(String[] args) {
        P1_BSTNode tree = new P1_BSTNode();
        int[] values = {10, 5, 9, 4, 7, 15, 20};

        for (int value : values) {
            tree.insert(value);
        }

        System.out.println(x:"Pre-order traversal:");
        tree.preOrder();
        System.out.println(x:"In-order traversal:");
        tree.inOrder();
        System.out.println(x:"Post-order traversal:");
        tree.postOrder();
    }
}
```

Output:

```
[Running] cd "a:\Programs\HTML & CSS (from Sems)\4th Semester\CSW-2\02-04-2025 [Chap-17] ASSIGNMENT-6\" && javac
Pre-order traversal:
10 5 4 9 7 15 20
In-order traversal:
4 5 7 9 10 15 20
Post-order traversal:
4 7 9 5 20 15 10
```

Q2. Construct a binary search tree from the given array of elements: {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}. Include a method called CreateTree to construct the binary search tree from a sorted array. This method takes an array of integers as input and constructs the tree recursively using a binary search algorithm.

Solution:

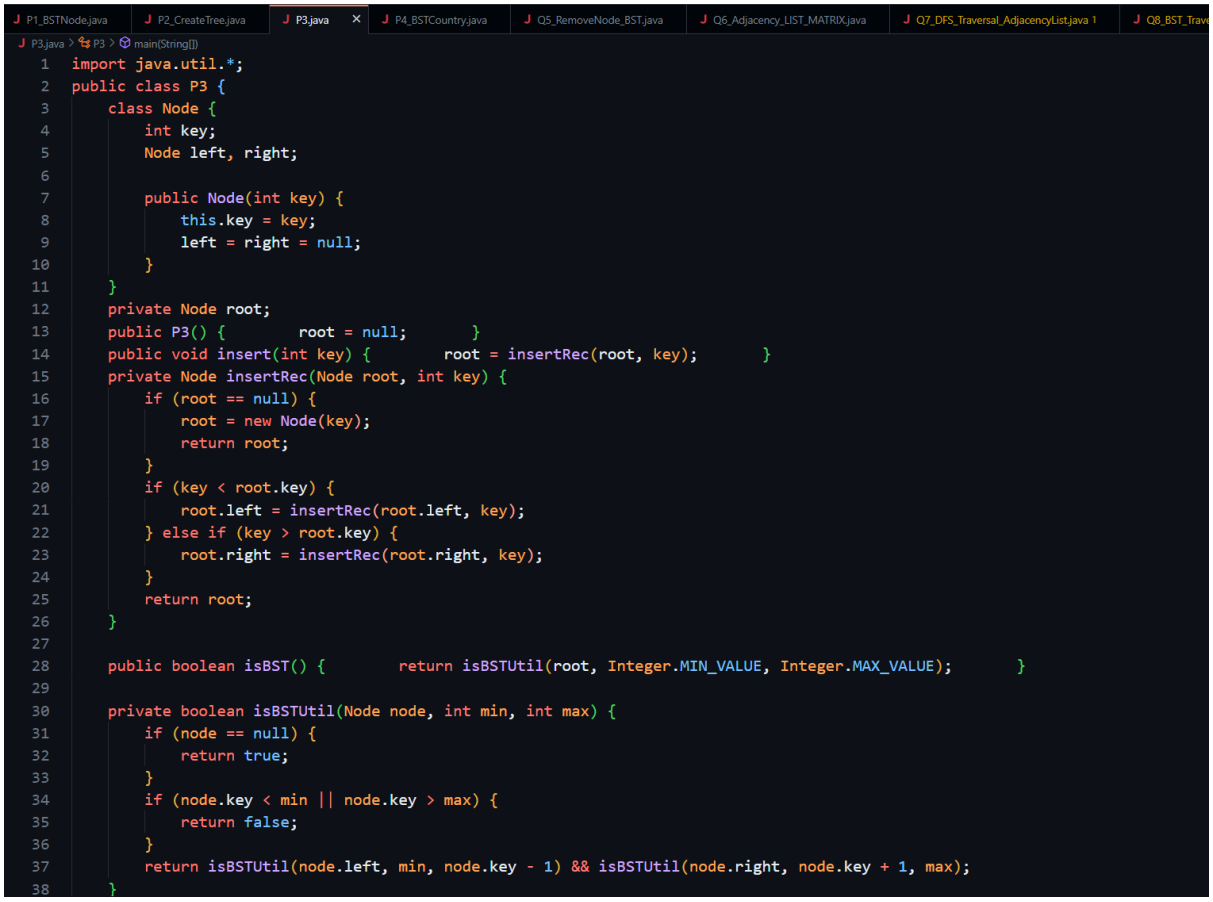
```java
class TreeNode {
    int value;
    TreeNode left, right;
    public TreeNode(int value) {
        this.value = value;
        left = right = null;
    }
}
class BST1 {
    private TreeNode root;
    public BST1() {
        root = null;
    }

    public void createTree(int[] sortedArr) {
        root = createTreeRec(sortedArr, start:0, sortedArr.length - 1);
    }
    private TreeNode createTreeRec(int[] sortedArr, int start, int end) {
        if (start > end) {
            return null;
        }

        int mid = (start + end) / 2;
        TreeNode node = new TreeNode(sortedArr[mid]);

        node.left = createTreeRec(sortedArr, start, mid - 1);
        node.right = createTreeRec(sortedArr, mid + 1, end);
        return node;
    }
    public void inorder(TreeNode node) {
        if (node != null) {
            inorder(node.left);
            System.out.print(node.value + " ");
            inorder(node.right);
        }
    }
    public TreeNode getRoot() {
        return root;
    }
}
public class P2_CreateTree {
    Run | Debug
    public static void main(String[] args) {
        int[] sortedArray = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
        BST1 bst = new BST1();
        bst.createTree(sortedArray);
        System.out.println(x:"Inorder Traversal of the constructed BST:");
        bst.inorder(bst.getRoot());
    }
}
```

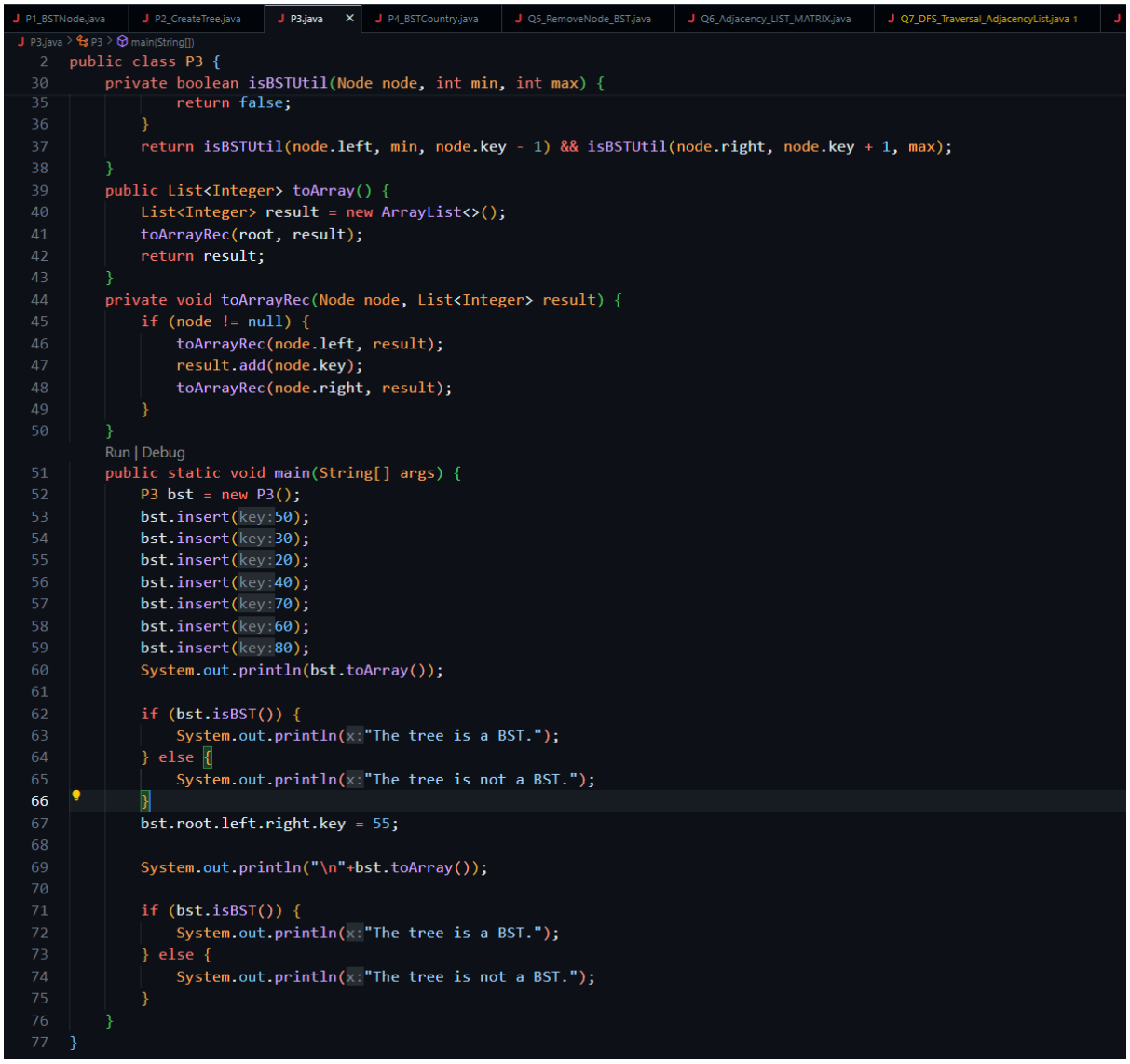
Output:

```
[Running] cd "a:\Programs\HTML & CSS (from Sems)\4th Semester\CSW-2\02-04-2025 [Chap-17] ASSIGNMENT-6\" && jav
Inorder Traversal of the constructed BST:
10 20 30 40 50 60 70 80 90 100
[Done] exited with code=0 in 0.992 seconds
```

Q3. Determine if a given binary tree is a binary search tree. You will use an isBST method, which takes the maximum and minimum range of the values of the nodes.

Solution:





Output:

```
[Running] cd "a:\Programs\HTML & CSS (from Sems)\4th Semester\CSW-2\02-04-2025 [Chap-17] ASSIGNMENT-6\" &
[20, 30, 40, 50, 60, 70, 80]
The tree is a BST.

[20, 30, 55, 50, 60, 70, 80]
The tree is not a BST.

[Done] exited with code=0 in 1.107 seconds
```

Q4. Design a Java program to manage country data using a binary search tree (BST). Create a class Country with members for name and population, along with a constructor and necessary methods. Define a class BNode to store a Country object and maintain references to its left and right children. Implement a class BSTCountry with a root node, a constructor, and a method to insert countries into the tree based on

their population. Extend the BST by adding methods for in-order traversal, finding the country with the maximum population (findMax), and finding the country with the minimum population (findMin). Finally, develop a main method to create a BST, insert country nodes, and invoke these methods to display the results.

Solution:

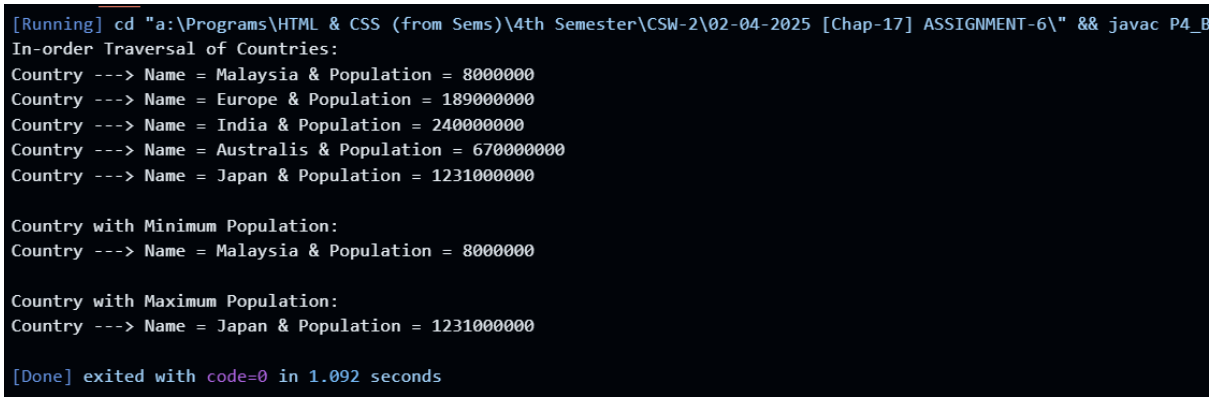
P1_BSTNode.java | P2_CreateTree.java | P3.java | P4_BSTCountry.java | Q5_RemoveNode_BST.java | Q6_Adjacency_LIST_MATRIX.java | Q7_DFS_Traversal_Adjace

P4_BSTCountry.java > P4_BSTCountry

```java
class Country {
    String name;
    int population;

    public Country(String name, int population) {
        this.name = name;
        this.population = population;
    }

    @Override
    public String toString() {
        return "Country ---> Name = " + name + " & Population = " + population;
    }
}

class BNode {
    Country country;
    BNode left, right;

    public BNode(Country country) {
        this.country = country;
        left = right = null;
    }
}

class BSTCountry {
    BNode rootNode;

    public BSTCountry() {
        rootNode = null;
    }

    public void insert(Country country) {
        rootNode = insertRec(rootNode, country);
    }

    private BNode insertRec(BNode root, Country country) {
        if (root == null) {
            return new BNode(country);
        }
        if (country.population < root.country.population) {
            root.left = insertRec(root.left, country);
        } else if (country.population > root.country.population) {
            root.right = insertRec(root.right, country);
        }
        return root;
```

```java
class BSTCountry {
    private BNode insertRec(BNode root, Country country) {
    }

    public void inOrderTraversal(BNode node) {
        if (node != null) {
            inOrderTraversal(node.left);
            System.out.println(node.country);
            inOrderTraversal(node.right);
        }
    }

    public Country findMax() {
        if (rootNode == null) {
            return null;
        }
        BNode current = rootNode;
        while (current.right != null) {
            current = current.right;
        }
        return current.country;
    }

    public Country findMin() {
        if (rootNode == null) {
            return null;
        }
        BNode current = rootNode;
        while (current.left != null) {
            current = current.left;
        }
        return current.country;
    }
}

public class P4_BSTCountry {
    Run | Debug
    public static void main(String[] args) {
        BSTCountry tree = new BSTCountry();

        tree.insert(new Country(name:"India", population:240000000));
        tree.insert(new Country(name:"Japan", population:1231000000));
        tree.insert(new Country(name:"Australis", population:670000000));
        tree.insert(new Country(name:"Europe", population:189000000));
        tree.insert(new Country(name:"Malaysia", population:8000000));

        System.out.println(x:"In-order Traversal of Countries:");
        tree.inOrderTraversal(tree.rootNode);

        System.out.println(x:"\nCountry with Minimum Population:");
        System.out.println(tree.findMin());

        System.out.println(x:"\nCountry with Maximum Population:");
        System.out.println(tree.findMax());
    }
}
```

Output:



Q5. Implement a method to remove node x from a binary search tree while ensuring that the tree maintains its properties. The deletion process involves three cases:

1. Case 1: Node x has no children (a leaf node).

2. Case 2: Node x has one child (either left or right).

3. Case 3: Node x has two children, requiring a suitable replacement to maintain the BST structure

Solution:

P1_BSTNode.java | P2_CreateTree.java | P3.java | P4_BSTCountry.java | Q5_RemoveNode_BST.java | Q6_Adjacency_LIST_MATRIX.java | Q7_DFS_Traversal_AdjacencyLi

Q5_RemoveNode_BST.java > ...

```java
class BSTNode {
    int info;
    BSTNode left, right;
    public BSTNode(int info) { this.info = info; }
}

class BST {
    BSTNode root;

    public void insert(int val) { root = insertRec(root, val); }
    private BSTNode insertRec(BSTNode node, int val) {
        if (node == null) return new BSTNode(val);
        if (val < node.info) node.left = insertRec(node.left, val);
        else if (val > node.info) node.right = insertRec(node.right, val);
        return node;
    }

    public void delete(int val) { root = deleteRec(root, val); }
    private BSTNode deleteRec(BSTNode node, int val) {
        if (node == null) return null;
        if (val < node.info) node.left = deleteRec(node.left, val);
        else if (val > node.info) node.right = deleteRec(node.right, val);
        else {
            if (node.left == null) return node.right;
            if (node.right == null) return node.left;
            BSTNode temp = minValue(node.right);
            node.info = temp.info;
            node.right = deleteRec(node.right, temp.info);
        }
        return node;
    }

    private BSTNode minValue(BSTNode node) {
        while (node.left != null) node = node.left;
        return node;
    }

    public void inorder(BSTNode node) {
        if (node != null) {
            inorder(node.left);
            System.out.print(node.info + " ");
            inorder(node.right);
        }
    }
}
```

Output:

```
[Running] cd "a:\Programs\HTML & CSS (from Sems)\4th Semester\CSW-2\02-04-2025 [Chap-1
Q5_RemoveNode_BST
BST Tree Created is: 20 30 40 50 60 70 80
Deletions completed. BST Tree after Deleting 20/30/50 is: 40 60 70 80
[Done] exited with code=0 in 1.028 seconds
```

Q6. Write a program to implement a graph using an adjacency matrix and adjacency list representation. Develop methods to construct the graph and display its adjacency matrix and adjacency list.

Solution:

```java
import java.util.*;

class Graph {
    private int V;
    private int[][] adjMatrix;
    private List<List<Integer>> adjList;

    public Graph(int V) {
        this.V = V;
        adjMatrix = new int[V][V];
        adjList = new ArrayList<>();
        for (int i = 0; i < V; i++) adjList.add(new ArrayList<>());
    }

    public void addEdge(int u, int v) {
        adjMatrix[u][v] = adjMatrix[v][u] = 1;
        adjList.get(u).add(v);
        adjList.get(v).add(u);
    }

    public void displayMatrix() {
        for (int[] row : adjMatrix) System.out.println(Arrays.toString(row));
    }

    public void displayList() {
        for (int i = 0; i < V; i++) System.out.println(i + " -> " + adjList.get(i));
    }
}

public class Q6_Adjacency_LIST_MATRIX{
    Run | Debug
    public static void main(String[] args) {
        Graph g = new Graph(V:5);
        g.addEdge(u:0, v:1); g.addEdge(u:0, v:4);
        g.addEdge(u:1, v:2); g.addEdge(u:1, v:3); g.addEdge(u:1, v:4);
        g.addEdge(u:2, v:3); g.addEdge(u:3, v:4);

        System.out.println(x:"Adjacency Matrix:");
        g.displayMatrix();

        System.out.println(x:"\nAdjacency List:");
        g.displayList();
    }
}
```

Output:

```
[Running] cd "a:\Programs\HTML & CSS (from Sems)\4th Semester\CSW-2\02-04-2025 [Chap-17] ASSIGNMENT-6\" &
Adjacency Matrix:
[0, 1, 0, 0, 1]
[1, 0, 1, 1, 1]
[0, 1, 0, 1, 0]
[0, 1, 1, 0, 1]
[1, 1, 0, 1, 0]

Adjacency List:
0 -> [1, 4]
1 -> [0, 2, 3, 4]
2 -> [1, 3]
3 -> [1, 2, 4]
4 -> [0, 1, 3]

[Done] exited with code=0 in 1.126 seconds
```

Q7. Create a class Graph that uses a linked list to represent N vertices. Implement a constructor to initialize the graph. Add a method to read a graph and store it using an adjacency list representation. Additionally, implement a Depth-First Search (DFS) method to traverse the graph's vertices. Finally, include a main method to create a graph, invoke the implemented methods, and display the traversal results.

Solution:

```java
import java.util.*;
class Graph1 {
    private int V;
    private LinkedList<Integer>[] adj;
    public Graph1(int V) {
        this.V = V;
        adj = new LinkedList[V];
        for (int i = 0; i < V; i++) adj[i] = new LinkedList<>();
    }

    public void addEdge(int u, int v) {
        adj[u].add(v);
        adj[v].add(u);
    }

    public void DFS(int start) {
        boolean[] visited = new boolean[V];
        dfsUtil(start, visited);
        System.out.println();
    }

    private void dfsUtil(int v, boolean[] visited) {
        visited[v] = true;
        System.out.print(v + " ");
        for (int neighbor : adj[v]) if (!visited[neighbor]) dfsUtil(neighbor, visited);
    }
}
public class Q7_DFS_Traversal_AdjacencyList{
    Run | Debug
    public static void main(String[] args) {
        Graph1 g = new Graph1(V:5);
        g.addEdge(u:0, v:1); g.addEdge(u:0, v:2);
        g.addEdge(u:1, v:3); g.addEdge(u:1, v:4);
        System.out.println(x:"DFS Traversal from node 0:");
        g.DFS(start:0);
    }
}
```

Output:

```
[Running] cd "a:\Programs\HTML & CSS (from Sems)\4th Semester\CSW-2\02-04-2025 [Chap-17] ASS
Q7_DFS_Traversal_AdjacencyList
Note: Q7_DFS_Traversal_AdjacencyList.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
DFS Traversal from node 0:
0 1 3 4 2

[Done] exited with code=0 in 1.853 seconds
```

Q8. Implement a Java program to traverse a graph using Breadth-First Search (BFS) with an adjacency list. Use ArrayDeque for efficient

traversal. The program should include methods to initialize the graph, add edges, display the adjacency list, and perform BFS. Finally, use the main method to construct the graph, invoke BFS, and display the traversal output.

Solution:

```java
import java.util.*;
class Graph2 {
    private int V;
    private List<Integer>[] adj;
    public Graph2(int V) {
        this.V = V;
        adj = new ArrayList[V];
        for (int i = 0; i < V; i++) adj[i] = new ArrayList<>();
    }
    public void addEdge(int u, int v) {
        adj[u].add(v);
        adj[v].add(u);
    }
    public void displayAdjList() {
        for (int i = 0; i < V; i++) System.out.println(i + " -> " + adj[i]);
    }
    public void BFS(int start) {
        boolean[] visited = new boolean[V];
        Queue<Integer> q = new ArrayDeque<>();
        visited[start] = true;
        q.add(start);

        while (!q.isEmpty()) {
            int v = q.poll();
            System.out.print(v + " ");
            for (int neighbor : adj[v]) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    q.add(neighbor);
                }
            }
        }
        System.out.println();
    }
}
public class Q8_BST_Traversal_AdjacencyList_ArrayDequeue{
    Run | Debug
    public static void main(String[] args) {
        Graph2 g = new Graph2(V:5);
        g.addEdge(u:0, v:1); g.addEdge(u:0, v:2);
        g.addEdge(u:1, v:3); g.addEdge(u:1, v:4);

        System.out.println(x:"Adjacency List:");
        g.displayAdjList();

        System.out.println(x:"\nBFS Traversal from node 0:");
        g.BFS(start:0);
    }
}
```

Output:

```
[Running] cd "a:\Programs\HTML & CSS (from Sems)\4th Semester\CSW-2\02-04-2025 [Chap-17] ASSIGNMENT-6\"
java && java Q8_BST_Traversal_AdjacencyList_ArrayDequeue
Note: Q8_BST_Traversal_AdjacencyList_ArrayDequeue.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Adjacency List:
0 -> [1, 2]
1 -> [0, 3, 4]
2 -> [0]
3 -> [1]
4 -> [1]

BFS Traversal from node 0:
0 1 2 3 4

[Done] exited with code=0 in 1.147 seconds
```