Name-Raghvi Sharma
Uid -22BCS17255
Section-611B

Assignment 6

1.Queue using Two Stacks

```
import java.util.Stack;

class QueueUsingStack {
    Stack<Integer> stack1 = new Stack<>();
    Stack<Integer> stack2 = new Stack<>();

    void enqueue(int data) {
        stack1.push(data);
    }

    int dequeue() {
        if (stack2.isEmpty()) {
            if (stack1.isEmpty()) {
                System.out.println("Queue is Empty");
                return -1;
            }

            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }

    void display() {
        if (stack1.isEmpty() && stack2.isEmpty()) {
            System.out.println("Queue is Empty");
            return;
        }
        Stack<Integer> temp = new Stack<>();
        temp.addAll(stack2);
        while (!temp.isEmpty()) {
            System.out.print(temp.pop() + " ");
        }
        for (int i = 0; i < stack1.size(); i++) {
            System.out.print(stack1.get(i) + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        QueueUsingStack queue = new QueueUsingStack();
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.display();

        System.out.println("Dequeued: " + queue.dequeue()); // Output: 10
        queue.display();

        queue.enqueue(40);
        queue.display();     }
}
```

2..Implement Deque using Stack

```
import java.util.Stack;
class QueueUsingStack {
    Stack<Integer> stack1 = new Stack<>();
    Stack<Integer> stack2 = new Stack<>();

    void enqueue(int data) {
        stack1.push(data);
    }

    int dequeue() {
        if (stack2.isEmpty()) {
            if (stack1.isEmpty()) {
                System.out.println("Queue is Empty");
                return -1;
            }

            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }

    void display() {
        if (stack1.isEmpty() && stack2.isEmpty()) {
            System.out.println("Queue is Empty");
            return;
        }
        Stack<Integer> temp = new Stack<>();
        temp.addAll(stack2);
        while (!temp.isEmpty()) {
            System.out.print(temp.pop() + " ");
        }
        for (int i = 0; i < stack1.size(); i++) {
            System.out.print(stack1.get(i) + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        QueueUsingStack queue = new QueueUsingStack();
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.display(); // Output: 10 20 30

        System.out.println("Dequeued: " + queue.dequeue()); // Output: 10
        queue.display(); // Output: 20 30

        queue.enqueue(40);
        queue.display(); // Output: 20 30 40
    }
}
```

3. Implement Stack using an Array

```
import java.util.Scanner;
```

```
class StackArray {
    int top;
    int maxSize;
    int[] stack;

    // Constructor
    StackArray(int size) {
        maxSize = size;
        stack = new int[maxSize];
        top = -1;
    }

    // Push operation
    void push(int data) {
        if (top == maxSize - 1) {
            System.out.println("Stack Overflow");
        } else {
            stack[++top] = data;
            System.out.println(data + " pushed to stack");
        }
    }

    // Pop operation
    int pop() {
        if (top == -1) {
            System.out.println("Stack Underflow");
            return -1;
        } else {
            return stack[top--];
        }
    }

    // Peek operation
    int peek() {
        if (top == -1) {
            System.out.println("Stack is Empty");
            return -1;
        } else {
            return stack[top];
        }
    }

    // Display operation
    void display() {
        if (top == -1) {
            System.out.println("Stack is Empty");
        } else {
            System.out.print("Stack elements: ");
            for (int i = 0; i <= top; i++) {
                System.out.print(stack[i] + " ");
            }
            System.out.println();
        }
    }

    // Main method
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        StackArray stack = new StackArray(5);

        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.display();  // Output: 10 20 30

        System.out.println("Popped: " + stack.pop());  // Output: 30
        stack.display();  // Output: 10 20

        System.out.println("Top element: " + stack.peek());  // Output: 20
    }
}
```

4. Implement Stack using Linked List

```
class StackLinkedList {
    // Node structure
    class Node {
        int data;
        Node next;
    }
```

```
Node top = null;  // Points to the top of the stack

// Push operation
void push(int data) {
    Node newNode = new Node();
    newNode.data = data;
    newNode.next = top;
    top = newNode;
    System.out.println(data + " pushed to stack");
}

// Pop operation
int pop() {
    if (top == null) {
        System.out.println("Stack Underflow");
        return -1;
    }
    int poppedData = top.data;
    top = top.next;
    return poppedData;
}

// Peek operation
int peek() {
    if (top == null) {
        System.out.println("Stack is Empty");
        return -1;
    }
    return top.data;
}

// Display operation
void display() {
    if (top == null) {
        System.out.println("Stack is Empty");
        return;
    }
    Node temp = top;
    System.out.print("Stack elements: ");
```

```
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }

    // Main method
    public static void main(String[] args) {
        StackLinkedList stack = new StackLinkedList();

        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.display();  // Output: 30 20 10

        System.out.println("Popped: " + stack.pop());  // Output: 30
        stack.display();  // Output: 20 10

        System.out.println("Top element: " + stack.peek());  // Output: 20
    }
}
```

5. Implement AVL Tree using BST

```
class AVLTree {
    class Node {
        int data, height;
        Node left, right;

        Node(int data) {
            this.data = data;
            height = 1; // New node is initially added at leaf
        }
    }

    Node root;
```

```
// Get height of node
int height(Node node) {
    if (node == null) return 0;
    return node.height;
}

// Get balance factor
int getBalance(Node node) {
    if (node == null) return 0;
    return height(node.left) - height(node.right);
}

// Right rotate
Node rightRotate(Node y) {
    Node x = y.left;
    Node T2 = x.right;

    // Perform rotation
    x.right = y;
    y.left = T2;

    // Update heights
    y.height = Math.max(height(y.left), height(y.right)) + 1;
    x.height = Math.max(height(x.left), height(x.right)) + 1;

    return x; // New root
}

Node leftRotate(Node x) {
    Node y = x.right;
    Node T2 = y.left;

    y.left = x;
    x.right = T2;

    x.height = Math.max(height(x.left), height(x.right)) + 1;
    y.height = Math.max(height(y.left), height(y.right)) + 1;

    return y; // New root
}
```

```
Node insert(Node node, int key) {

    if (node == null) return new Node(key);
    if (key < node.data) node.left = insert(node.left, key);
    else if (key > node.data) node.right = insert(node.right, key);

    node.height = 1 + Math.max(height(node.left), height(node.right));

    int balance = getBalance(node);

    if (balance > 1 && key < node.left.data) return rightRotate(node);


    if (balance < -1 && key > node.right.data) return leftRotate(node);

    if (balance > 1 && key > node.left.data) {
        node.left = leftRotate(node.left);
        return rightRotate(node);
    }

    if (balance < -1 && key < node.right.data) {
        node.right = rightRotate(node.right);
        return leftRotate(node);
    }

    return node;
}

void inorder(Node node) {
    if (node != null) {
        inorder(node.left);
        System.out.print(node.data + " ");
        inorder(node.right);
    }
}

public static void main(String[] args) {
    AVLTree tree = new AVLTree();

    tree.root = tree.insert(tree.root, 30);
```

```
        tree.root = tree.insert(tree.root, 20);
        tree.root = tree.insert(tree.root, 40);
        tree.root = tree.insert(tree.root, 10);
        tree.root = tree.insert(tree.root, 25);
        tree.root = tree.insert(tree.root, 50);

        System.out.print("In-order Traversal: ");
        tree.inorder(tree.root);  // Balanced output
    }
}
```

6. Implement Topological Sorting using Graph + Stack (DFS)

```
import java.util.*;

class Graph {
    private int V;  // Number of vertices
    private List<List<Integer>> adj;  // Adjacency list

    Graph(int V) {
        this.V = V;
        adj = new ArrayList<>();
        for (int i = 0; i < V; i++)
            adj.add(new ArrayList<>());
    }

    void addEdge(int u, int v) {
        adj.get(u).add(v);
    }


    void dfs(int node, boolean[] visited, Stack<Integer> stack) {
        visited[node] = true;
        for (int neighbor : adj.get(node)) {
            if (!visited[neighbor])
                dfs(neighbor, visited, stack);
        }
        stack.push(node); // After visiting neighbors, push to stack
    }

    void topologicalSort() {
        Stack<Integer> stack = new Stack<>();
        boolean[] visited = new boolean[V];

        // Call DFS for unvisited nodes
        for (int i = 0; i < V; i++) {
            if (!visited[i])
                dfs(i, visited, stack);
        }


        System.out.print("Topological Sort: ");
        while (!stack.isEmpty())
            System.out.print(stack.pop() + " ");
    }

    public static void main(String[] args) {
        Graph g = new Graph(6);
        g.addEdge(5, 0);
        g.addEdge(5, 2);
        g.addEdge(4, 0);
        g.addEdge(4, 1);
        g.addEdge(2, 3);
        g.addEdge(3, 1);

        g.topologicalSort();
    }
}
```