GALGOTIAS UNIVERSITY

DSA File

Name – Md Arshad Iqubal

Admission number – 23SCSE1010461

Subject - Data Structures using JAVA

Code - R1UC303B

Section – 28

Faculty – DR Vikash Yogendra Mishra sir

Program 1: Addition and Multiplication of Two 2D Arrays

```
import java.util.Arrays;
public class ArrayOperations {
    public static void main(String[] args) {
        int[][] A = {{1, 2}, {3, 4}};
        int[][] B = {{5, 6}, {7, 8}};

        // Addition
        int[][] addition = new int[A.length][A[0].length];
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < A[0].length; j++) {
                addition[i][j] = A[i][j] + B[i][j];
            }
        }

        // Multiplication
        int[][] multiplication = new int[A.length][B[0].length];
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < B[0].length; j++) {
                for (int k = 0; k < A[0].length; k++) {
                    multiplication[i][j] += A[i][k] * B[k][j];
                }
            }
        }

        System.out.println("Addition of matrices: " + Arrays.deepToString(addition));
        System.out.println("Multiplication of matrices: " + Arrays.deepToString(multiplication));
    }
}
```

Output:

```
Addition of matrices: [[6, 8], [10, 12]]
Multiplication of matrices: [[19, 22], [43, 50]]
```

Program 2: Linear Search and Binary Search

```
import java.util.Arrays;
class Search {
    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i;
            }
        }
        return -1;
    }
    public static int binarySearch(int[] arr, int target) {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == target) {
                return mid;
            } else if (arr[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 7, 9};
        System.out.println("Linear Search Index: " + linearSearch(arr, 5));
        System.out.println("Binary Search Index: " + binarySearch(arr, 5));
    }
}
```

Output :

```
Linear Search Index: 2
Binary Search Index: 2
```

Program 3: Write a program to implement Insertion Sort.

```
class InsertionSort {
   public static void insertionSort(int[] arr) {
      for (int i = 1; i < arr.length; i++) {
         int key = arr[i];
         int j = i - 1;
         while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
         }
         arr[j + 1] = key;
      }
   }
   public static void main(String[] args) {
      int[] arr = {5, 3, 1, 9, 8};
      insertionSort(arr);
      System.out.println("Insertion Sort: " + Arrays.toString(arr));
   }
}
```

Output:

```
Insertion Sort: [1, 3, 5, 8, 9]
```

Program 4: Write a program to implement Bubble Sort .

```
class BubbleSort {
   public static void bubbleSort(int[] arr) {
      int n = arr.length;
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
               int temp = arr[j];
               arr[j] = arr[j + 1];
               arr[j + 1] = temp;
            }
```

```
        }
      }
    }

    public static void main(String[] args) {
      int[] arr = {5, 3, 1, 9, 8};
      bubbleSort(arr);
      System.out.println("Bubble Sort: " + Arrays.toString(arr));
    }
}
```

Output:

```
[Running] cd "d:\scilab\" && javac BubbleSort.java && java BubbleSort
Bubble Sort: [1, 3, 5, 8, 9]
```

Program 5: Write a program to implement Singly Linked List.

```
class SinglyLinkedList {
    static class Node {
      int data;
      Node next;

      Node(int data) {
        this.data = data;
        this.next = null;
      }
    }
    private Node head;
    public void insert(int data) {
      Node newNode = new Node(data);
      if (head == null) {
        head = newNode;
      } else {
        Node temp = head;
        while (temp.next != null) {
          temp = temp.next;
```

```
            }
            temp.next = newNode;
        }
    }

    public void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        SinglyLinkedList sll = new SinglyLinkedList();
        sll.insert(1);
        sll.insert(2);
        sll.insert(3);
        sll.display();
    }
}
```

Output:

```
[Running] cd "d:\scilab\" && javac SinglyLinkedList.java && java SinglyLinkedList
1 -> 2 -> 3 -> null
```

Program 6: Write a program to implement stack using array.

```
class StackArray {

    private int[] stack;
    private int top;
    private int capacity;

    public StackArray(int size) {
```

```
        stack = new int[size];
        top = -1;
        capacity = size;
    }

    public void push(int value) {
        if (top == capacity - 1) {
            System.out.println("Stack Overflow");
            return;
        }
        stack[++top] = value;
    }

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

    public int peek() {
        if (top == -1) {
            System.out.println("Stack is empty");
            return -1;
        }
        return stack[top];
    }

    public void display() {
        for (int i = top; i >= 0; i--) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
```

```
    }

    public static void main(String[] args) {
        StackArray stack = new StackArray(5);
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.display();
        System.out.println("Popped: " + stack.pop());
        System.out.println("Peek: " + stack.peek());
        stack.display();
    }
}
```

Output:

```
[Running] cd "d:\scilab\" && javac StackArray.java && java StackArray
30 20 10
Popped: 30
Peek: 20
20 10
```

Program 7: Write a program to implement queue using array.

```
class QueueArray {
    private int[] queue;
    private int front;
    private int rear;
    private int capacity;
    private int size;
    public QueueArray(int capacity) {
        this.capacity = capacity;
        queue = new int[capacity];
        front = 0;
        rear = -1;
        size = 0;
    }
```

```
public void enqueue(int value) {
    if (size == capacity) {
        System.out.println("Queue Overflow");
        return;
    }
    rear = (rear + 1) % capacity;
    queue[rear] = value;
    size++;
}

public int dequeue() {
    if (size == 0) {
        System.out.println("Queue Underflow");
        return -1;
    }
    int value = queue[front];
    front = (front + 1) % capacity;
    size--;
    return value;
}

public void display() {
    for (int i = 0; i < size; i++) {
        System.out.print(queue[(front + i) % capacity] + " ");
    }
 System.out.println();
}

public static void main(String[] args) {
    QueueArray queue = new QueueArray(5);
    queue.enqueue(10);
    queue.enqueue(20);
    queue.enqueue(30);
    queue.display();
```

```
        System.out.println("Dequeued: " + queue.dequeue());
        queue.display();
    }
}
```

Output:

```
[Running] cd "d:\scilab\" && javac QueueArray.java && java QueueArray
10 20 30
Dequeued: 10
20 30
```

Program 8: Write a program to implement circular queue using array.

```
class CircularQueue {
    private int[] queue;
    private int front;
    private int rear;
    private int size;
    private int capacity;

    public CircularQueue(int capacity) {
        this.capacity = capacity;
        queue = new int[capacity];
        front = -1;
        rear = -1;
        size = 0;
    }
    public void enqueue(int value) {
        if (size == capacity) {
            System.out.println("Queue Overflow");
            return;
        }
        if (front == -1) {
            front = 0;
        }
        rear = (rear + 1) % capacity;
        queue[rear] = value;
```

```
    size++;
}

public int dequeue() {
    if (size == 0) {
        System.out.println("Queue Underflow");
        return -1;
    }
    int value = queue[front];
    front = (front + 1) % capacity;
    size--;
    if (size == 0) {
        front = -1;
        rear = -1;
    }
    return value;
}

public void display() {
    if (size == 0) {
        System.out.println("Queue is empty");
        return;
    }
    for (int i = 0; i < size; i++) {
        System.out.print(queue[(front + i) % capacity] + " ");
    }
    System.out.println();
}

public static void main(String[] args) {
    CircularQueue cq = new CircularQueue(5);
    cq.enqueue(10);
    cq.enqueue(20);
    cq.enqueue(30);
```

```
        cq.display();
        System.out.println("Dequeued: " + cq.dequeue());
        cq.display();
    }
}
```

Output:

```
[Running] cd "d:\scilab\" && javac CircularQueue.java && java CircularQueue
10 20 30
Dequeued: 10
20 30
```

Program 9: Write a program to implement stack using linked list.

```
class StackLinkedList {
    private static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
        }
    }
    private Node top;
    public void push(int value) {
        Node newNode = new Node(value);
     newNode.next = top;
        top = newNode;
    }

    public int pop() {
        if (top == null) {
            System.out.println("Stack Underflow");
            return -1;
        }
        int value = top.data;
        top = top.next;
        return value;
```

```
    }

    public int peek() {
        if (top == null) {
            System.out.println("Stack is empty");
            return -1;
        }
        return top.data;
    }

    public void display() {
        Node temp = top;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        StackLinkedList stack = new StackLinkedList();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.display();
        System.out.println("Popped: " + stack.pop());
        System.out.println("Peek: " + stack.peek());
        stack.display();
    }
}
```

Output:

```
[Running] cd "d:\scilab\" && javac StackLinkedList.java && java StackLinkedList
30 -> 20 -> 10 -> null
Popped: 30
Peek: 20
20 -> 10 -> null
```

Program 10: Write a program to implement queue using linked list.

```
class QueueLinkedList {
    private static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
        }
    }

    private Node front;
    private Node rear;

    public void enqueue(int value) {
        Node newNode = new Node(value);
        if (rear == null) {
            front = rear = newNode;
        } else {
            rear.next = newNode;
            rear = newNode;
        }
    }

    public int dequeue() {
        if (front == null) {
            System.out.println("Queue Underflow");
            return -1;
        }
```

```
        int value = front.data;
        front = front.next;
        if (front == null) {
            rear = null;
        }
        return value;
    }

    public void display() {
        Node temp = front;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        QueueLinkedList queue = new QueueLinkedList();
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.display();
        System.out.println("Dequeued: " + queue.dequeue());
        queue.display();
    }
}
```

Output:

```
[Running] cd "d:\scilab\" && javac QueueLinkedList.java && java QueueLinkedList
10 -> 20 -> 30 -> null
Dequeued: 10
20 -> 30 -> null
```

Program 11: Write a program to implement circular queue using linked list.

```
class CircularQueueLinkedList {
    private static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
        }
    }

    private Node front;
    private Node rear;

    public void enqueue(int value) {
        Node newNode = new Node(value);
        if (front == null) {
            front = rear = newNode;
            rear.next = front;
        } else {
            rear.next = newNode;
            rear = newNode;
          rear.next = front;
        }
    }

    public int dequeue() {
        if (front == null) {
            System.out.println("Queue Underflow");
            return -1;
        }
        int value = front.data;
        if (front == rear) {
            front = rear = null;
        } else {
```

```
            front = front.next;
            rear.next = front;
        }
        return value;
    }

    public void display() {
        if (front == null) {
            System.out.println("Queue is empty");
            return;
        }
        Node temp = front;
        do {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        } while (temp != front);
        System.out.println("(back to front)");
    }

    public static void main(String[] args) {
      CircularQueueLinkedList cq = new CircularQueueLinkedList();
        cq.enqueue(10);
        cq.enqueue(20);
        cq.enqueue(30);
        cq.display();
        System.out.println("Dequeued: " + cq.dequeue());
        cq.display();
    }
}
```

Output:

```
[Running] cd "d:\scilab\" && javac CircularQueueLinkedList.java && java CircularQueueLinkedList
10 -> 20 -> 30 -> (back to front)
Dequeued: 10
20 -> 30 -> (back to front)
```

Program 12: Write a program to implement binary search tree using linked list.

```
class BinarySearchTree {
   private static class Node {
      int data;
      Node left, right;

      Node(int data) {
         this.data = data;
         left = right = null;
      }
   }

   private Node root;

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

   private Node insertRec(Node root, int value) {
   if (root == null) {
         root = new Node(value);
         return root;
      }
      if (value < root.data) {
         root.left = insertRec(root.left, value);
      } else if (value > root.data) {
         root.right = insertRec(root.right, value);
      }
      return root;
   }

   public void inorder() {
      inorderRec(root);
      System.out.println();
```

```
    }

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

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        bst.insert(50);
        bst.insert(30);
        bst.insert(70);
        bst.insert(20);
        bst.insert(40);
        bst.insert(60);
     bst.insert(80);
        System.out.println("Inorder Traversal: ");
        bst.inorder();
    }
}
```

Output:

```
[Running] cd "d:\scilab\" && javac BinarySearchTree.java && java BinarySearchTree
Inorder Traversal:
20 30 40 50 60 70 80
```