Task 1:

Create  a custom node , add elements to it and traverse it..

public class task01 {

    // Custom Node

    static class Node {

        int data;

        Node next;

        Node(int data) {

            this.data = data;

            next = null;

        }

    }

    static class MyList {

        Node head;

        MyList() {

            head = null;

        }

        void addElement(int data) {

            Node newNode = new Node(data);

            if (head == null) {

                head = newNode;

                return;

            }

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

        }

        void traverseList() {

            if (head == null) {

                System.out.println("List is empty!");

                return;

            }

            Node current = head;

            System.out.print("Elements in list: ");

            while (current != null) {

                System.out.print(current.data + " ");

                current = current.next;

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        MyList list = new MyList();

        System.out.println("Adding elements: 5, 10, 15, 20");

        list.addElement(5);

        list.addElement(10);

        list.addElement(15);

        list.addElement(20);

        list.traverseList();

        System.out.println("\nAdding more elements: 25, 30");

        list.addElement(25);

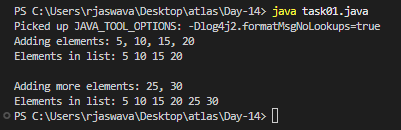
        list.addElement(30);

        list.traverseList();

    }

}

Output:



Task 2:

What do you understand by traversing a linked list?

Traversing always start from the first node, visits each node exactly once stops when it reaches the end (null).

Task 3:

Create a Circular Linked list using Task 1 Singly linked list/ doubly linked list.

public class task03 {

    static class Node {

        int data;

        Node next;

        Node(int data) {

            this.data = data;

            next = null;

        }

    }

    static class CircularLinkedList {

        Node head;

        CircularLinkedList() {

            head = null;

        }

        void addElement(int data) {

            Node newNode = new Node(data);

            if (head == null) {

                head = newNode;

                head.next = head;

                return;

            }

            Node current = head;

            while (current.next != head) {

                current = current.next;

            }

            current.next = newNode;

            newNode.next = head;

        }

        void traverseList() {

            if (head == null) {

                System.out.println("List is empty!");

                return;

            }

            Node current = head;

            System.out.print("Elements in circular list: ");

            do {

                System.out.print(current.data + " ");

                current = current.next;

            } while (current != head);

            System.out.println();

        }

        void showMultipleRounds(int rounds) {

            if (head == null) {

                System.out.println("List is empty!");

                return;

            }

            System.out.print("Multiple rounds: ");

            Node current = head;

            int roundCount = 0;

            do {

                System.out.print(current.data + " ");

                current = current.next;

                if (current == head) {

                    roundCount++;

                    if (roundCount < rounds) {

                        System.out.print("| ");

                    }

                }

            } while (roundCount < rounds);

            System.out.println();

        }

    }

    public static void main(String[] args) {

        CircularLinkedList list = new CircularLinkedList();

        System.out.println("Adding elements: 5, 10, 15, 20, 25");

        list.addElement(5);

        list.addElement(10);

        list.addElement(15);

        list.addElement(20);

        list.addElement(25);

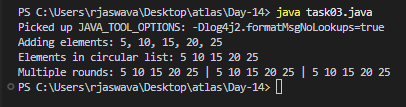
        list.traverseList();

        list.showMultipleRounds(3);

    }

}

Output:



Task 4:

Delete a node in the circular linked list

public class task04 {

    static class Node {

        int data;

        Node next;

        Node(int data) {

            this.data = data;

            next = null;

        }

    }

    static class CircularLinkedList {

        Node head;

        CircularLinkedList() {

            head = null;

        }

        void addElement(int data) {

            Node newNode = new Node(data);

            if (head == null) {

                head = newNode;

                head.next = head;

                return;

            }

            Node current = head;

            while (current.next != head) {

                current = current.next;

            }

            current.next = newNode;

            newNode.next = head;

        }

        void deleteNode(int value) {

            if (head == null) {

                System.out.println("List is empty! Nothing to delete.");

                return;

            }

            Node current = head;

            Node prev = null;

            if (current.data == value) {

                if (current.next == head) {

                    head = null;

                    System.out.println("Deleted " + value + " (was the only node)");

                    return;

                }

                while (current.next != head) {

                    current = current.next;

                }

                current.next = head.next;

                head = head.next;

                System.out.println("Deleted " + value + " (was head node)");

                return;

            }

            do {

                prev = current;

                current = current.next;

                if (current.data == value) {

                    prev.next = current.next;

                    System.out.println("Deleted " + value);

                    return;

                }

            } while (current != head);

            System.out.println("Value " + value + " not found in the list!");

        }

        void traverseList() {

            if (head == null) {

                System.out.println("List is empty!");

                return;

            }

            Node current = head;

            System.out.print("Elements in circular list: ");

            do {

                System.out.print(current.data + " ");

                current = current.next;

            } while (current != head);

            System.out.println();

        }

        void showMultipleRounds(int rounds) {

            if (head == null) {

                System.out.println("List is empty!");

                return;

            }

            System.out.print("Multiple rounds: ");

            Node current = head;

            int roundCount = 0;

            do {

                System.out.print(current.data + " ");

                current = current.next;

                if (current == head) {

                    roundCount++;

                    if (roundCount < rounds) {

                        System.out.print("| ");

                    }

                }

            } while (roundCount < rounds);

            System.out.println();

        }

    }

    public static void main(String[] args) {

        CircularLinkedList list = new CircularLinkedList();

        System.out.println("Adding elements: 5, 10, 15, 20, 25");

        list.addElement(5);

        list.addElement(10);

        list.addElement(15);

        list.addElement(20);

        list.addElement(25);

        System.out.println("\nOriginal List:");

        list.traverseList();

        list.showMultipleRounds(2);

        System.out.println("\nDeleting 15...");

        list.deleteNode(15);

        list.traverseList();

        list.showMultipleRounds(2);

        System.out.println("\nDeleting 5...");

        list.deleteNode(5);

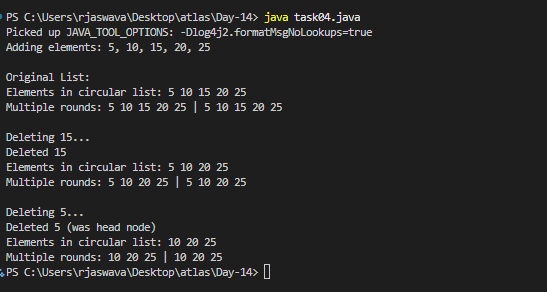
        list.traverseList();

        list.showMultipleRounds(2);

    }

}

Output:



Task 5:

Create  a  stack and pop the element also print the popped element.

public class task05 {

    public static void main(String[] args) {

        int[] stack = new int[5];

        int top = -1;

        System.out.println("Adding elements to stack:");

        top++;

        stack[top] = 10;

        System.out.println("Added: " + stack[top]);

        top++;

        stack[top] = 20;

        System.out.println("Added: " + stack[top]);

        top++;

        stack[top] = 30;

        System.out.println("Added: " + stack[top]);

        System.out.println("\nCurrent stack:");

        for(int i = top; i >= 0; i--) {

            System.out.println(stack[i]);

        }

        System.out.println("\nRemoving elements from stack:");

        System.out.println("Removed: " + stack[top]);

        top--;

        System.out.println("Removed: " + stack[top]);

        top--;

        System.out.println("\nRemaining stack:");

        for(int i = top; i >= 0; i--) {

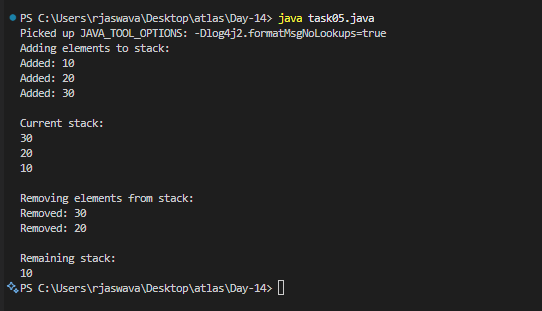
            System.out.println(stack[i]);

        }

    }

}

Output:



Task 6:

Find an element in the stack and display the position

Hint 👍

Int position = names.search(“value”);

public class task06 {

    public static void main(String[] args) {

        int[] stack = new int[5];

        int top = -1;

        System.out.println("Adding elements to stack:");

        top++;

        stack[top] = 10;

        System.out.println("Added: " + stack[top]);

        top++;

        stack[top] = 20;

        System.out.println("Added: " + stack[top]);

        top++;

        stack[top] = 30;

        System.out.println("Added: " + stack[top]);

        System.out.println("\nCurrent stack:");

        for(int i = top; i >= 0; i--) {

            System.out.println("Position " + i + ": " + stack[i]);

        }

        int searchValue = 20;

        System.out.println("\nSearching for value: " + searchValue);

        boolean found = false;

        for(int i = top; i >= 0; i--) {

            if(stack[i] == searchValue) {

                System.out.println("Found " + searchValue + " at position: " + i);

                System.out.println("Distance from top: " + (top - i));

                found = true;

                break;

            }

        }

        if(!found) {

            System.out.println(searchValue + " not found in stack");

        }

        searchValue = 20;

        System.out.println("\nSearching for value: " + searchValue);

        found = false;

        for(int i = top; i >= 0; i--) {

            if(stack[i] == searchValue) {

                System.out.println("Found " + searchValue + " at position: " + i);

                System.out.println("Distance from top: " + (top - i));

                found = true;

                break;

            }

        }

        if(!found) {

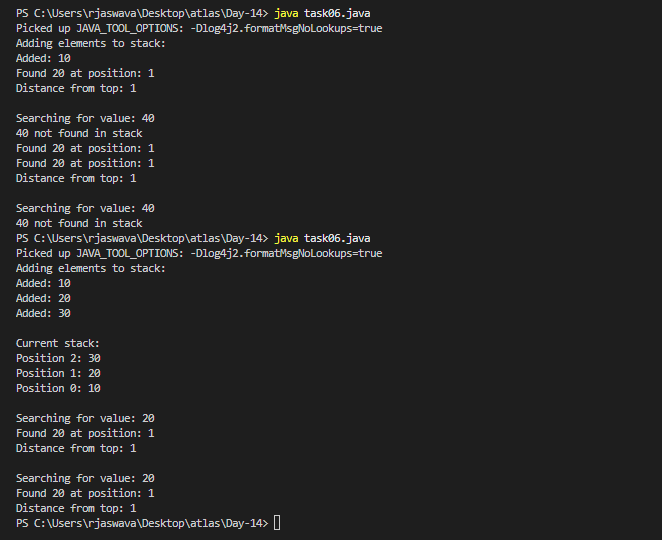
            System.out.println(searchValue + " not found in stack");

        }

    }

}

Output:



Task 7:

Peek the element and print it ..

public class task07 {

    public static void main(String[] args) {

        int[] stack = new int[5];

        int top = -1;

        top++;

        stack[top] = 10;

        top++;

        stack[top] = 20;

        top++;

        stack[top] = 30;

        System.out.println("Peeking at top element...");

        if(top >= 0) {

            System.out.println("Top element is: " + stack[top]);

        } else {

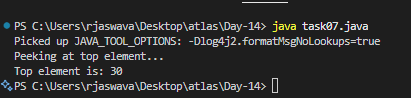
            System.out.println("Stack is empty!");

        }

    }

}

Output:



Task 8:

Check if the stack is empty or not?

public class task08 {

    public static void main(String[] args) {

        int[] stack = new int[5];

        int top = -1;

        System.out.println("Is stack empty initially?");

        if(top == -1) {

            System.out.println("Yes, stack is empty");

        } else {

            System.out.println("No, stack has elements");

        }

        System.out.println("\nAdding elements...");

        top++;

        stack[top] = 10;

        top++;

        stack[top] = 20;

        System.out.println("\nIs stack empty after adding elements?");

        if(top == -1) {

            System.out.println("Yes, stack is empty");

        } else {

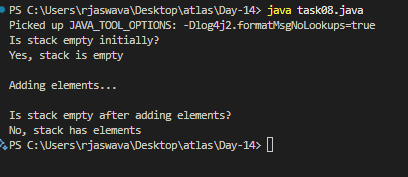
            System.out.println("No, stack has elements");

        }

    }

}

Output:



Task 9:

What are the methods of the stack class.. List them down.. With a one liner..

* Push() – adds element to top of stack
* Pop() – Removes and returns top element
* Peek() – Returns top element without removing
* Empty() – checks if stack is empty
* Search() – returns position of element
* Size() – Returns number of elements in stacks
* Clear() – Remove all elements from stack
* Contains() – Check if element exists in stack.

Task 10:

Wap to create  a queue with custom methods

Is empty ()

Is full()

Enque

Deque

Peek

display()

public class task10 {

    static class Queue {

        int[] arr;

        int front, rear;

        int size;

        Queue(int size) {

            this.size = size;

            arr = new int[size];

            front = rear = -1;

        }

        boolean isEmpty() {

            return front == -1;

        }

        boolean isFull() {

            return (rear + 1) % size == front;

        }

        void enqueue(int data) {

            if(isFull()) {

                System.out.println("Queue is full!");

                return;

            }

            if(isEmpty()) front = 0;

            rear = (rear + 1) % size;

            arr[rear] = data;

            System.out.println("Added: " + data);

        }

        void dequeue() {

            if(isEmpty()) {

                System.out.println("Queue is empty!");

                return;

            }

            System.out.println("Removed: " + arr[front]);

            if(front == rear) {

                front = rear = -1;

            } else {

                front = (front + 1) % size;

            }

        }

        void peek() {

            if(isEmpty()) {

                System.out.println("Queue is empty!");

                return;

            }

            System.out.println("Front element: " + arr[front]);

        }

        void display() {

            if(isEmpty()) {

                System.out.println("Queue is empty!");

                return;

            }

            System.out.print("Queue: ");

            int i = front;

            do {

                System.out.print(arr[i] + " ");

                i = (i + 1) % size;

            } while(i != (rear + 1) % size);

            System.out.println();

        }

    }

    public static void main(String[] args) {

        Queue q = new Queue(3);

        q.enqueue(10);

        q.enqueue(20);

        q.display();

        q.peek();

        q.dequeue();

        q.display();

    }

}

Output:

