**1.Pseudo**

**Start**

**Input**: Size of the array (size).

**Initialize**: Array arr with size size + 10.

**Input**: Array elements for indices 0 to size - 1.

**Repeat** until the user exits:

1. **Display Menu**:
   * 1: Insert an element.
   * 2: Delete an element.
   * 3: Find maximum and minimum.
   * 4: Display the array.
   * 5: Exit.
2. **Input**: User's choice (choice).
3. **Switch** (choice):
   * **Case 1**: Insert an element
     1. Input the element to insert (element).
     2. Input the position (position).
     3. If position < 1 or position > size + 1, display "Invalid position".
     4. Else:
        + Shift elements from index size - 1 to position - 1 by one position to the right.
        + Insert element at position - 1.
        + Increment size.
        + Display "Element inserted".
   * **Case 2**: Delete an element
     1. Input the position to delete (position).
     2. If position < 1 or position > size, display "Invalid position".
     3. Else:
        + Shift elements from index position to size - 1 by one position to the left.
        + Decrement size.
        + Display "Element deleted".
   * **Case 3**: Find maximum and minimum
     1. If size == 0, display "Array is empty".
     2. Else:
        + Initialize max and min to arr[0].
        + Loop through the array from index 1 to size - 1:
          - Update max if arr[i] > max.
          - Update min if arr[i] < min.
        + Display max and min.
   * **Case 4**: Display the array
     1. If size == 0, display "Array is empty".
     2. Else:
        + Loop through the array from index 0 to size - 1 and display elements.
   * **Case 5**: Exit
     1. Display "Exiting..." and terminate the program.
   * **Default**: Display "Invalid choice".

**End**.

**2.Psuedo**

1. **Start**
2. **Define** a structure Node with:
   * data
   * Pointer next (initially null).
3. **Initialize**:
   * head as null.
4. **Repeat** until the user exits:
   * **Display Menu**:
     + 1: Insert at beginning.
     + 2: Insert at end.
     + 3: Delete from beginning.
     + 4: Delete from end.
     + 5: Display the list.
     + 6: Exit.
   * **Input**: User's choice (choice).
5. **Switch** (choice):
   * **Case 1**: Insert at beginning
     + Input data.
     + Create a new node newNode with data.
     + Set newNode.next = head.
     + Update head = newNode.
     + Display "Node inserted at beginning".
   * **Case 2**: Insert at end
     + Input data.
     + Create a new node newNode with data.
     + If head == null:
       - Set head = newNode.
     + Else:
       - Initialize temp = head.
       - Traverse to the last node (temp.next != null).
       - Set temp.next = newNode.
     + Display "Node inserted at end".
   * **Case 3**: Delete from beginning
     + If head == null, display "List is empty".
     + Else:
       - Set temp = head.
       - Update head = head.next.
       - Free temp.
       - Display "Node deleted from beginning".
   * **Case 4**: Delete from end
     + If head == null, display "List is empty".
     + Else if head.next == null:
       - Free head.
       - Set head = null.
     + Else:
       - Initialize temp = head.
       - Traverse to the second last node (temp.next.next != null).
       - Free temp.next.
       - Set temp.next = null.
       - Display "Node deleted from end".
   * **Case 5**: Display the list
     + If head == null, display "List is empty".
     + Else:
       - Initialize temp = head.
       - While temp != null:
         * Print temp.data.
         * Move temp = temp.next.
   * **Case 6**: Exit
     + Display "Exiting..." and terminate the program.
   * **Default**: Display "Invalid choice".
6. **End**.

**Program**

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public class SinglyLinkedList {

Node head;

void insertAtBeginning(int data) {

Node newNode = new Node(data);

newNode.next = head;

head = newNode;

System.out.println("Node inserted at the beginning.");

}

void insertAtEnd(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;

}

System.out.println("Node inserted at the end.");

}

void deleteFromBeginning() {

if (head == null) {

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

} else {

head = head.next;

System.out.println("Node deleted from the beginning.");

}

}

void deleteFromEnd() {

if (head == null) {

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

} else if (head.next == null) {

head = null;

System.out.println("Node deleted from the end.");

} else {

Node temp = head;

while (temp.next.next != null) {

temp = temp.next;

}

temp.next = null;

System.out.println("Node deleted from the end.");

}

}

void displayList() {

if (head == null) {

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

} else {

Node temp = head;

System.out.print("Linked List: ");

while (temp != null) {

System.out.print(temp.data + " -> ");

temp = temp.next;

}

System.out.println("null");

}

}

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nChoose an operation:");

System.out.println("1. Insert at beginning");

System.out.println("2. Insert at end");

System.out.println("3. Delete from beginning");

System.out.println("4. Delete from end");

System.out.println("5. Display list");

System.out.println("6. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter data to insert: ");

int dataBegin = scanner.nextInt();

list.insertAtBeginning(dataBegin);

break;

case 2:

System.out.print("Enter data to insert: ");

int dataEnd = scanner.nextInt();

list.insertAtEnd(dataEnd);

break;

case 3:

list.deleteFromBeginning();

break;

case 4:

list.deleteFromEnd();

break;

case 5:

list.displayList();

break;

case 6:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid choice. Try again.");

}

}

}

}

**3.Program**

import java.util.Scanner;

class DoublyNode {

int data;

DoublyNode prev;

DoublyNode next;

DoublyNode(int data) {

this.data = data;

this.prev = null;

this.next = null;

}

}

public class DoublyLinkedList {

DoublyNode head;

void insertAtBeginning(int data) {

DoublyNode newNode = new DoublyNode(data);

if (head != null) {

newNode.next = head;

head.prev = newNode;

}

head = newNode;

System.out.println("Node inserted at the beginning.");

}

void insertAtEnd(int data) {

DoublyNode newNode = new DoublyNode(data);

if (head == null) {

head = newNode;

} else {

DoublyNode temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

newNode.prev = temp;

}

System.out.println("Node inserted at the end.");

}

void insertAtPosition(int data, int position) {

if (position <= 0) {

System.out.println("Invalid position.");

return;

}

DoublyNode newNode = new DoublyNode(data);

if (position == 1) {

insertAtBeginning(data);

return;

}

DoublyNode temp = head;

for (int i = 1; temp != null && i < position - 1; i++) {

temp = temp.next;

}

if (temp == null) {

System.out.println("Position out of bounds.");

return;

}

newNode.next = temp.next;

if (temp.next != null) {

temp.next.prev = newNode;

}

temp.next = newNode;

newNode.prev = temp;

System.out.println("Node inserted at position " + position + ".");

}

void deleteFromBeginning() {

if (head == null) {

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

} else {

head = head.next;

if (head != null) {

head.prev = null;

}

System.out.println("Node deleted from the beginning.");

}

}

void deleteFromEnd() {

if (head == null) {

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

} else if (head.next == null) {

head = null;

System.out.println("Node deleted from the end.");

} else {

DoublyNode temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.prev.next = null;

System.out.println("Node deleted from the end.");

}

}

void deleteFromPosition(int position) {

if (position <= 0 || head == null) {

System.out.println("Invalid position or list is empty.");

return;

}

if (position == 1) {

deleteFromBeginning();

return;

}

DoublyNode temp = head;

for (int i = 1; temp != null && i < position; i++) {

temp = temp.next;

}

if (temp == null) {

System.out.println("Position out of bounds.");

return;

}

if (temp.next != null) {

temp.next.prev = temp.prev;

}

if (temp.prev != null) {

temp.prev.next = temp.next;

}

System.out.println("Node deleted from position " + position + ".");

}

void displayList() {

if (head == null) {

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

} else {

DoublyNode temp = head;

System.out.print("Doubly Linked List: ");

while (temp != null) {

System.out.print(temp.data + " <-> ");

temp = temp.next;

}

System.out.println("null");

}

}

public static void main(String[] args) {

DoublyLinkedList list = new DoublyLinkedList();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nChoose an operation:");

System.out.println("1. Insert at beginning");

System.out.println("2. Insert at end");

System.out.println("3. Insert at specific position");

System.out.println("4. Delete from beginning");

System.out.println("5. Delete from end");

System.out.println("6. Delete from specific position");

System.out.println("7. Display list");

System.out.println("8. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter data to insert: ");

int dataBegin = scanner.nextInt();

list.insertAtBeginning(dataBegin);

break;

case 2:

System.out.print("Enter data to insert: ");

int dataEnd = scanner.nextInt();

list.insertAtEnd(dataEnd);

break;

case 3:

System.out.print("Enter data to insert: ");

int dataPos = scanner.nextInt();

System.out.print("Enter position: ");

int position = scanner.nextInt();

list.insertAtPosition(dataPos, position);

break;

case 4:

list.deleteFromBeginning();

break;

case 5:

list.deleteFromEnd();

break;

case 6:

System.out.print("Enter position to delete: ");

int delPos = scanner.nextInt();

list.deleteFromPosition(delPos);

break;

case 7:

list.displayList();

break;

case 8:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid choice. Try again.");

}

}

}

}

**Pseudo**

Define Node class:

Attributes:

data: Integer

prev: Reference to previous node (initially NULL)

next: Reference to next node (initially NULL)

Define DoublyLinkedList class:

Attribute:

head: Reference to the first node (initially NULL)

Function insertAtBeginning(data):

Create a new node with data

If head is NULL:

Set head to new node

Else:

Set newNode.next to head

Set head.prev to newNode

Update head to newNode

Print "Node inserted at the beginning"

Function insertAtEnd(data):

Create a new node with data

If head is NULL:

Set head to new node

Else:

Initialize temp to head

While temp.next is not NULL:

Move temp to temp.next

Set temp.next to newNode

Set newNode.prev to temp

Print "Node inserted at the end"

Function insertAtPosition(data, position):

If position <= 0:

Print "Invalid position"

Return

Create a new node with data

If position == 1:

Call insertAtBeginning(data)

Return

Initialize temp to head

For i = 1 to position - 2:

If temp is NULL:

Print "Position out of bounds"

Return

Move temp to temp.next

Set newNode.next to temp.next

If temp.next is not NULL:

Set temp.next.prev to newNode

Set temp.next to newNode

Set newNode.prev to temp

Print "Node inserted at position"

Function deleteFromBeginning():

If head is NULL:

Print "List is empty"

Return

Set head to head.next

If head is not NULL:

Set head.prev to NULL

Print "Node deleted from the beginning"

Function deleteFromEnd():

If head is NULL:

Print "List is empty"

Return

If head.next is NULL:

Set head to NULL

Else:

Initialize temp to head

While temp.next is not NULL:

Move temp to temp.next

Set temp.prev.next to NULL

Print "Node deleted from the end"

Function deleteFromPosition(position):

If position <= 0 OR head is NULL:

Print "Invalid position or list is empty"

Return

If position == 1:

Call deleteFromBeginning()

Return

Initialize temp to head

For i = 1 to position - 1:

If temp is NULL:

Print "Position out of bounds"

Return

Move temp to temp.next

If temp.next is not NULL:

Set temp.next.prev to temp.prev

If temp.prev is not NULL:

Set temp.prev.next to temp.next

Print "Node deleted from position"

Function displayList():

If head is NULL:

Print "List is empty"

Return

Initialize temp to head

Print "Doubly Linked List: "

While temp is not NULL:

Print temp.data + " <-> "

Move temp to temp.next

Print "NULL"

Main Function:

Initialize list as a DoublyLinkedList

While true:

Print menu with operations:

1. Insert at beginning

2. Insert at end

3. Insert at specific position

4. Delete from beginning

5. Delete from end

6. Delete from specific position

7. Display list

8. Exit

Read user choice

Switch(choice):

Case 1:

Read data

Call insertAtBeginning(data)

Case 2:

Read data

Call insertAtEnd(data)

Case 3:

Read data and position

Call insertAtPosition(data, position)

Case 4:

Call deleteFromBeginning()

Case 5:

Call deleteFromEnd()

Case 6:

Read position

Call deleteFromPosition(position)

Case 7:

Call displayList()

Case 8:

Print "Exiting..."

Exit loop

Default:

Print "Invalid choice"

End

**4.Pseudo**

Define Stack class:

Attributes:

stack: An empty list to store elements

maxSize: Maximum size of the stack (optional)

Methods:

push(element):

If stack size is less than maxSize (if defined):

Add element to stack

Print "Element pushed onto stack"

Else:

Print "Stack overflow"

pop():

If stack is not empty:

Remove and return the last element

Print "Element popped from stack"

Else:

Print "Stack underflow"

peek():

If stack is not empty:

Return the last element

Print "Top element is: <element>"

Else:

Print "Stack is empty"

display():

If stack is not empty:

Print "Stack: " followed by stack elements

Else:

Print "Stack is empty"

Main Function:

Initialize stack as an instance of Stack

While true:

Print menu with operations:

1. Push

2. Pop

3. Peek

4. Display

5. Exit

Read user choice

Switch(choice):

Case 1:

Read element to push

Call stack.push(element)

Case 2:

Call stack.pop()

Case 3:

Call stack.peek()

Case 4:

Call stack.display()

Case 5:

Print "Exiting..."

Exit loop

Default:

Print "Invalid choice"

**Program**

import java.util.Scanner;

class Stack {

private int[] stack;

private int top;

private int maxSize;

// Constructor to initialize the stack

public Stack(int size) {

maxSize = size;

stack = new int[maxSize];

top = -1;

}

// Push operation

public void push(int element) {

if (top == maxSize - 1) {

System.out.println("Stack Overflow! Cannot add more elements.");

} else {

stack[++top] = element;

System.out.println(element + " pushed onto the stack.");

}

}

// Pop operation

public void pop() {

if (top == -1) {

System.out.println("Stack Underflow! No elements to pop.");

} else {

System.out.println(stack[top] + " popped from the stack.");

top--;

}

}

// Peek operation

public void peek() {

if (top == -1) {

System.out.println("Stack is empty! No top element.");

} else {

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

}

}

// Display stack elements

public 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();

}

}

}

public class StackImplementation {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the stack: ");

int size = scanner.nextInt();

Stack stack = new Stack(size);

while (true) {

System.out.println("\nChoose an operation:");

System.out.println("1. Push");

System.out.println("2. Pop");

System.out.println("3. Peek");

System.out.println("4. Display");

System.out.println("5. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter the element to push: ");

int element = scanner.nextInt();

stack.push(element);

break;

case 2:

stack.pop();

break;

case 3:

stack.peek();

break;

case 4:

stack.display();

break;

case 5:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid choice! Please try again.");

}

}

}

}