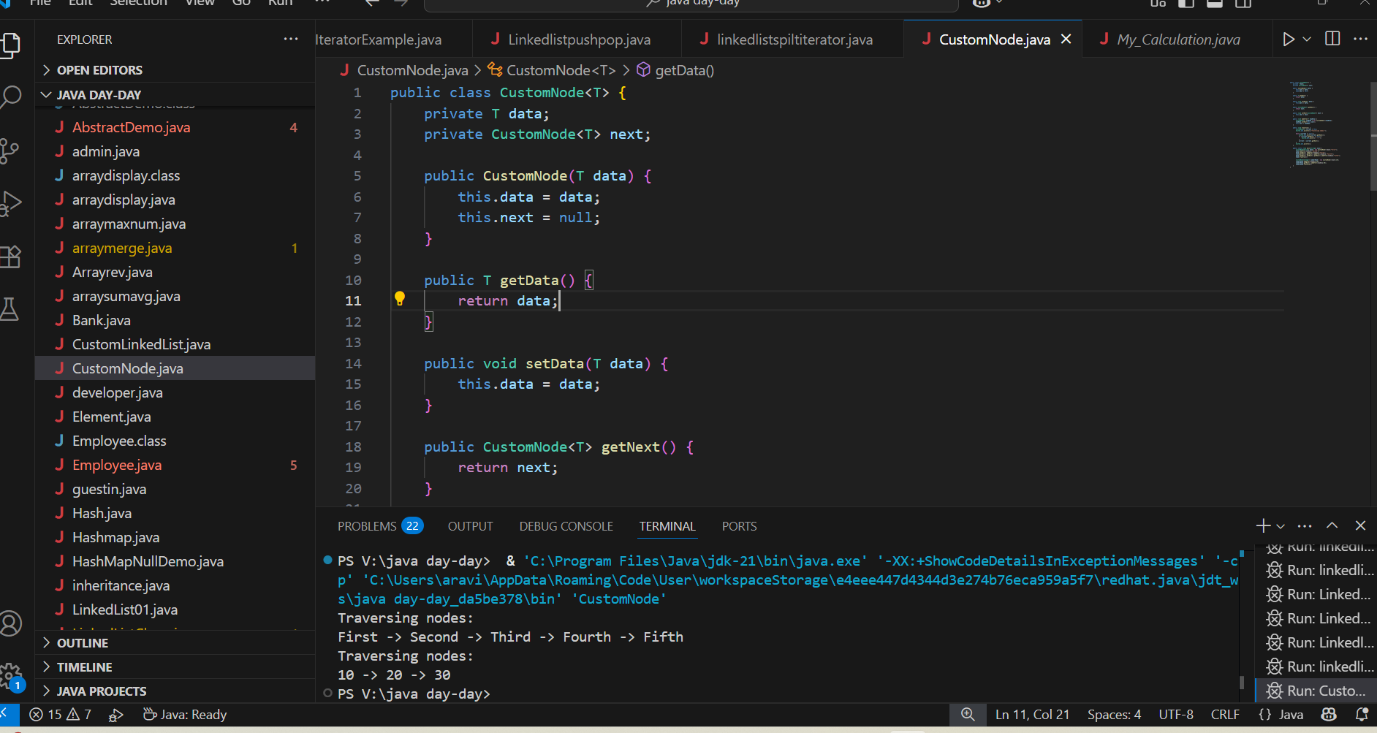
Day 14 - 02th July 2025

Name: Aravind Kasanagottu

Id: mvsnarav

Task 1:

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



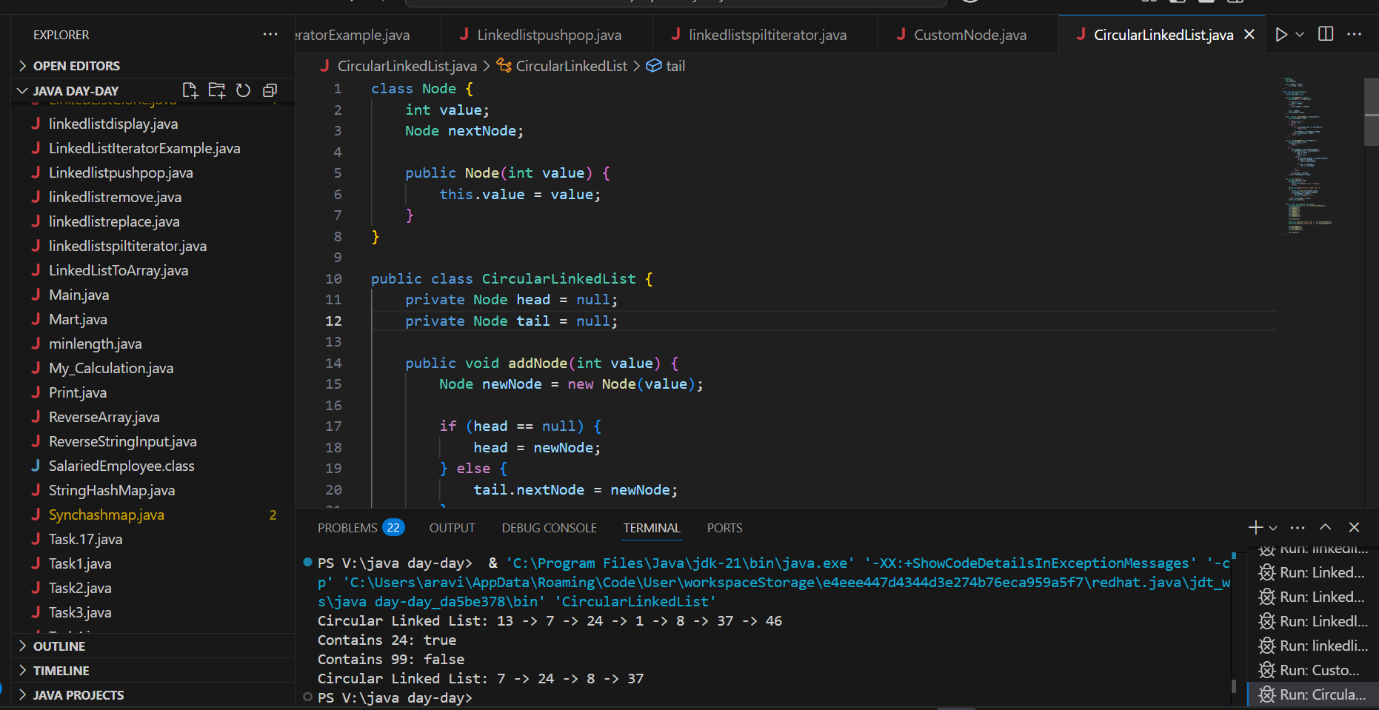
Task 2 👍

What do you understand by traversing elements in a linked list.

Traversing a linked list means visiting each node one by one, starting from the head, until you reach the end (null).

Task 3:

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



class Node {

    int value;

    Node nextNode;

    public Node(int value) {

        this.value = value;

    }

}

public class CircularLinkedList {

// if the list is empty

    private Node head = null;

    private Node tail = null;

//   …..

}

public void addNode(int value) {

    Node newNode = new Node(value);

    if (head == null) {

        head = newNode;

    } else {

        tail.nextNode = newNode;

    }

    tail = newNode;

    tail.nextNode = head;

}

private CircularLinkedList createCircularLinkedList() {

    CircularLinkedList cll = new CircularLinkedList();

    cll.addNode(13);

    cll.addNode(7);

    cll.addNode(24);

    cll.addNode(1);

    cll.addNode(8);

    cll.addNode(37);

    cll.addNode(46);

    return cll;

}

public boolean containsNode(int searchValue) {

    Node currentNode = head;

    if (head == null) {

        return false;

    } else {

        do {

            if (currentNode.value == searchValue) {

                return true;

            }

            currentNode = currentNode.nextNode;

        } while (currentNode != head);

        return false;

    }

}

public void deleteNode(int valueToDelete) {

    Node currentNode = head;

    if (head == null) { // the list is empty

        return;

    }

    do {

        Node nextNode = currentNode.nextNode;

        if (nextNode.value == valueToDelete) {

            if (tail == head) { // the list has only one single element

                head = null;

                tail = null;

            } else {

                currentNode.nextNode = nextNode.nextNode;

                if (head == nextNode) { //we're deleting the head

                    head = head.nextNode;

                }

                if (tail == nextNode) { //we're deleting the tail

                    tail = currentNode;

                }

            }

            break;

        }

        currentNode = nextNode;

    } while (currentNode != head);

}

Stacks

Collection (interface ) ===> List (interface) ⇒ Vector class ===> Stacks (class)

Task 4 :

Stacks

Create  a code to implement a stack

import java.util.Stack;

class Ds\_Stack\_Push {

    public static void main(String[] args) {

        Stack<String> names= new Stack<>();

        names.push("Prasunamba");

        names.push("Meher");

        names.push(".MK");

        System.out.println("Stack of names: " + names);

    }

}

Task 5:

Create  astack and pop the element also print the popped element.

12.50 plz complete

import java.util.Stack;

class Ds\_Stack\_Pop {

    public static void main(String[] args) {

        Stack<String> names= new Stack<>();

        names.push("Prasunamba");

        names.push("Meher");

        names.push(".MK");

System.out.println("before deletion ");

        System.out.println("Stack of names: " + names);

System.out.println("after deletion ");

String dummy = names.pop();

System.out.println("Stack of names: " + names);

System.out.println("deleted element is "+ dummy);

    }

}

Task 6:

Find an element in the stack and display the position

Hint 👍

Int position = names.search(“value”);

import java.util.Stack;

class Ds\_Stack\_SearchPosition  {

    public static void main(String[] args) {

        Stack<String> names= new Stack<>();

        names.push("Prasunamba");

        names.push("Meher");

        names.push(".MK");

        System.out.println("Stack of names: " + names);

String Val = "Meher";

int position = names.search(Val);

System.out.println("the searched value is at position  " + position);

    }

}

Task 7:

Peek the element and print it ..

import java.util.Stack;

class Ds\_Stack\_Peek  {

    public static void main(String[] args) {

        Stack<String> names= new Stack<>();

        names.push("Prasunamba");

        names.push("Meher");

        names.push(".MK");

        System.out.println("Stack of names: " + names);

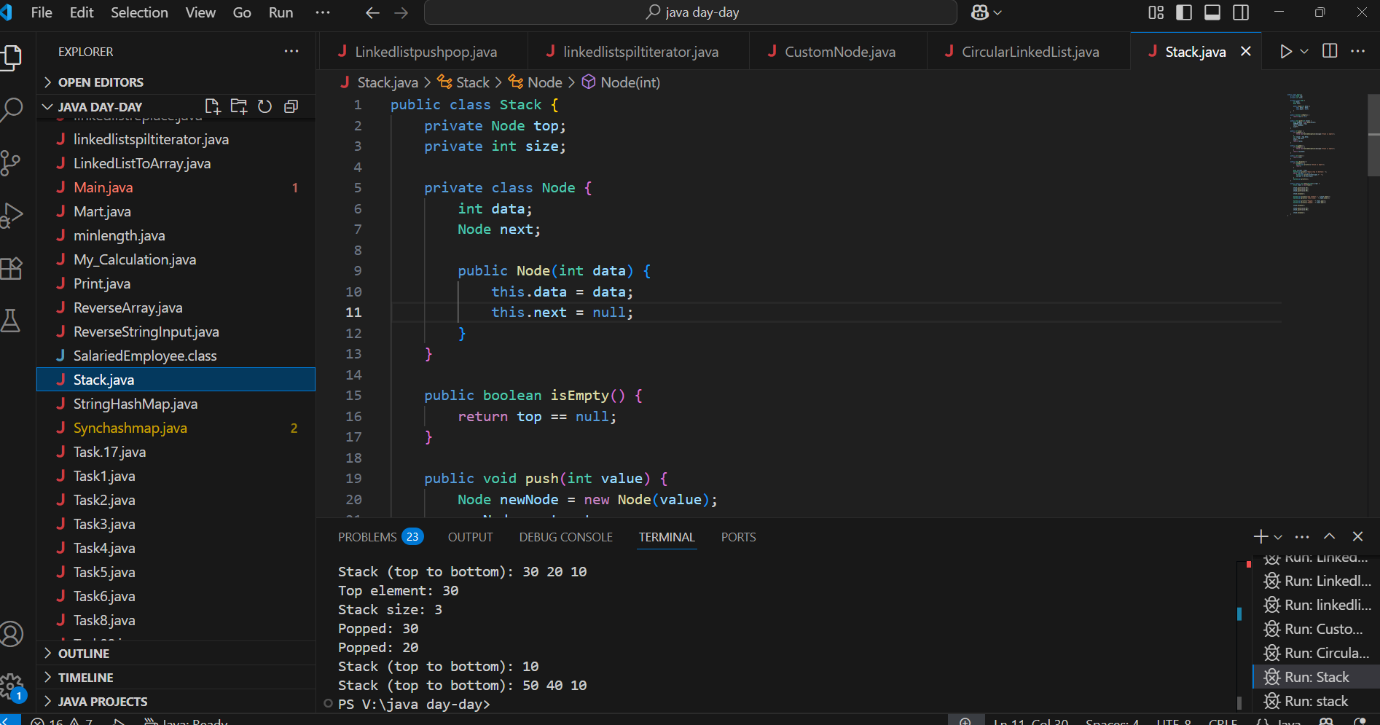
String topElement = names.peek();

System.out.println("The top element (without removal); "+ topElement);

System.out.println("Stack of names after peek operation " + names);

    }

}



THE ABOVE SNIPPET FOR TASK 4-8

Task 8:

Check if the stack is empty or not?

import java.util.Stack;

public class Ds\_Stack\_Empty {

public static void main(String[] args) {

         Stack<String> names = new Stack<>();

        System.out.println("Is the stack empty? " + names.empty());

names.push("Prasunamba");

             names.push("Meher");

             System.out.println("Is the stack empty? " + names.empty());

      while (!names.empty()){

                 System.out.println("Popped: " + names.pop());

             }

                System.out.println("Is the stack empty? " + names.empty());

}

}

Task 9:

What are the methods of the stack class

Core Stack Methods:

1. push(int value)
   * Adds a new element to the *top* of the stack
   * Example: stack.push(5) adds 5 to the stack
2. pop()
   * Removes and returns the *top* element
   * Throws an error if stack is empty
   * Example: int val = stack.pop()
3. peek()
   * Returns the *top* element without removing it
   * Throws an error if stack is empty
   * Example: int topVal = stack.peek()

Helper Methods:

1. isEmpty()
   * Checks if stack has no elements
   * Returns true if empty, false otherwise
   * Example: if(stack.isEmpty())
2. size()
   * Returns current number of elements in stack
   * Example: int count = stack.size()
3. display()
   * Prints all elements from top to bottom
   * Example: stack.display() prints "Stack (top to bottom): 3 2 1"

Supporting Structure:

* Node**(inner class)**
  + Private class that stores:
    - data (the integer value)
    - next (reference to next node)
  + Used internally to implement the stack

Key Characteristics:

* Follows **LIFO** (Last-In-First-Out) principle
* Uses **linked nodes** (no array size limit)
* All operations are **O(1)** except display() which is O(n)
* Includes **error handling** for empty stack cases

Queues

Task 10:

What are the common operations in Queues

**1. Core Queue Operations**

| **Operation** | **Description** | **Time Complexity** | **Example** |
| --- | --- | --- | --- |
| enqueue() (or add()) | Inserts an element at the **rear** (end) of the queue | *O(1)* | queue.enqueue(5) |
| dequeue() (or remove()) | Removes and returns the element from the **front** of the queue | *O(1)* | int val = queue.dequeue() |
| peek() (or front()) | Returns the **front element** without removing it | *O(1)* | int first = queue.peek() |

**2. Helper/Utility Operations**

| **Operation** | **Description** | **Time Complexity** | **Example** |
| --- | --- | --- | --- |
| isEmpty() | Checks if the queue is empty | *O(1)* | if (queue.isEmpty()) |
| isFull() | Checks if the queue is full (for fixed-size queues) | *O(1)* | if (queue.isFull()) |
| size() | Returns the number of elements in the queue | *O(1)* | int count = queue.size() |
| display() | Prints all elements (front to rear) | *O(n)* | queue.display() |

**3. Special Cases (Varies by Implementation)**

* **Priority Queue**:
  + insert() (adds elements based on priority)
  + extractMax() or extractMin()
* **Circular Queue**:
  + Handles wrap-around for efficient space usage.
* **Double-Ended Queue (Deque)**:
  + addFirst(), addLast(), removeFirst(), removeLast()

**Key Properties of Queues**

1. **FIFO (First-In-First-Out)**: The first element added is the first to be removed.
2. **Two Pointers**:
   * **Front** (for deletion)
   * **Rear** (for insertion)
3. **Implementation Choices**:
   * **Array-based** (fixed size, efficient but may need resizing)
   * **Linked List-based** (dynamic size, no capacity issues)

Task 11:

Wap to create  a queue with custom methods

Is empty ()

Is full()

Enque

Deque

Peek

display()

public class Queue {

int queueLength = 3;

   int items[] = new int[queueLength];

   int front = -1;

   int back = -1;

void enQueue(int itemValue) {

if(isFull()){

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

       } else if(front == -1 && back == -1){

           front = back = 0;

           items[back] = itemValue;

       } else{

           back++;

           items[back] = itemValue;

       }

   }

   void deQueue(){

       if(isEmpty()){

           System.out.println("Queue is empty. Nothing to dequeue");

       } else if (front == back){

         front = back = -1;

       } else {

         front++;

       }

   }

   void display(){

       int i;

if(isEmpty()){

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

       } else {

           for(i = front; i <= back; i++){

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

           }

       }

}

boolean isFull(){

       if(back == queueLength - 1){

           return true;

       } else {

           return false;

       }

   }

boolean isEmpty(){

       if(front == -1 && back == -1){

           return true;

       } else {

           return false;

       }

   }

   void peek(){

       System.out.println("Front value is: " + items[front]);

   }

public static void main(String[] args) {

     Queue myQueue = new Queue();

myQueue.enQueue(111);

     myQueue.enQueue(222);

     myQueue.enQueue(777);

     myQueue.display();

myQueue.peek();

}

}

Add ons:

//converting stack and deque into a lists and printing their elements in java using streams.

import java.util.\*;

import java.util.stream.Collectors;

class Stack\_Deque\_to\_List.java {

    public static void main (String[] args) {

          Stack<Integer> stack = new Stack<>();

        Deque<Integer> deque = new ArrayDeque<>();

        stack.push(1);

        deque.push(1);

        stack.push(2);

        deque.push(2);

        List<Integer> list1 = stack.stream().collect(Collectors.toList());

          System.out.println("Using Stack: ");

          for(int i = 0; i < list1.size(); i++){

              System.out.print(list1.get(i) + " " );

        }

          System.out.println();

        List<Integer> list2 = deque.stream().collect(Collectors.toList());

          System.out.println("Using Deque: ");

          for(int i = 0; i < list2.size(); i++){

              System.out.print(list2.get(i) + " " );

        }

          System.out.println();

    }

}

==========================================================================================================================================