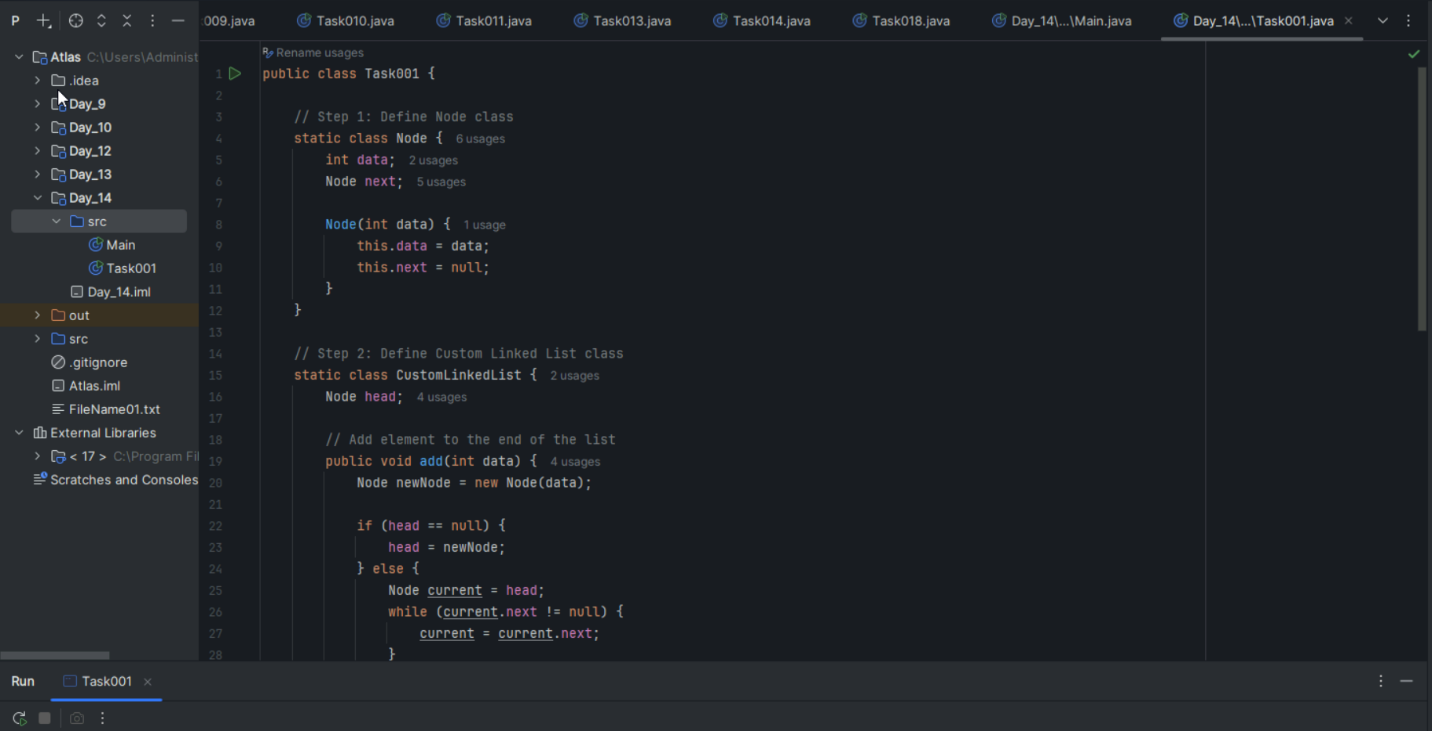
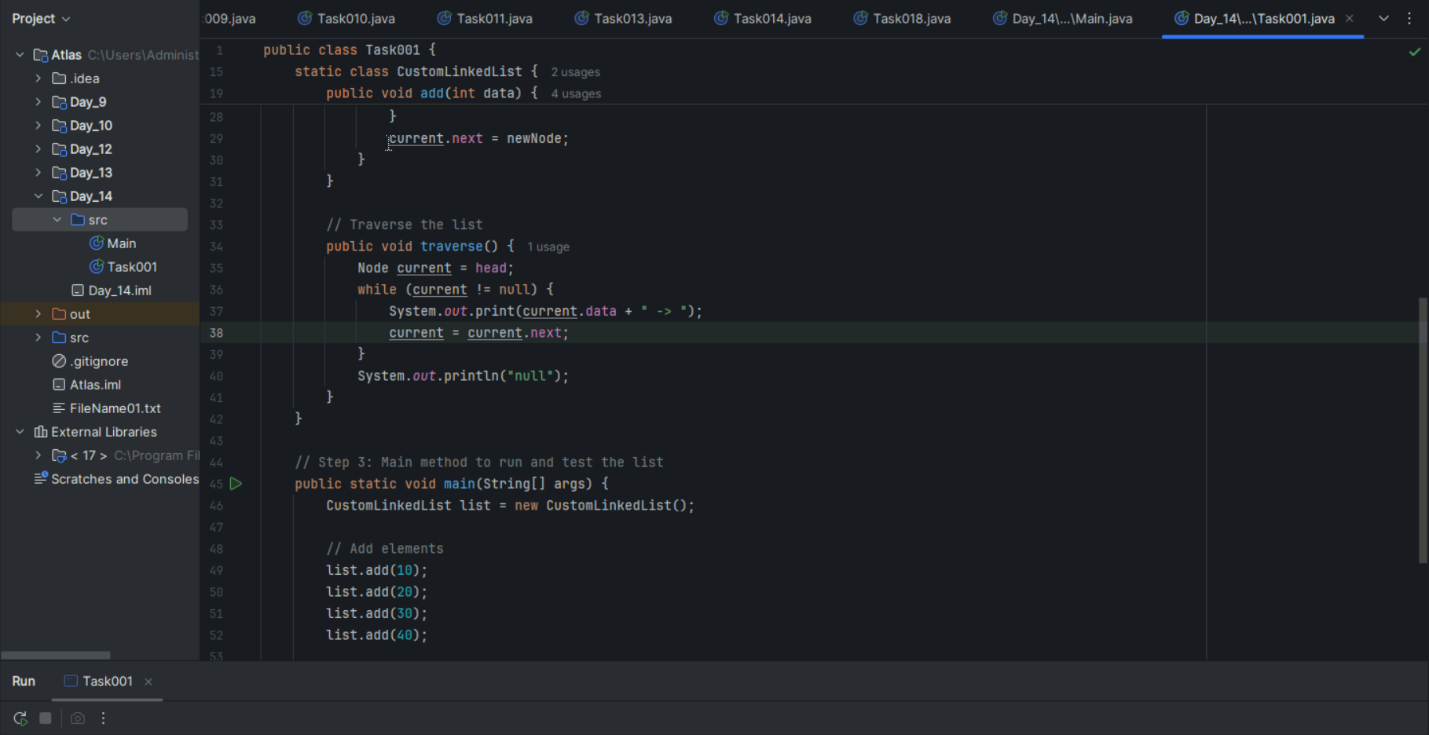
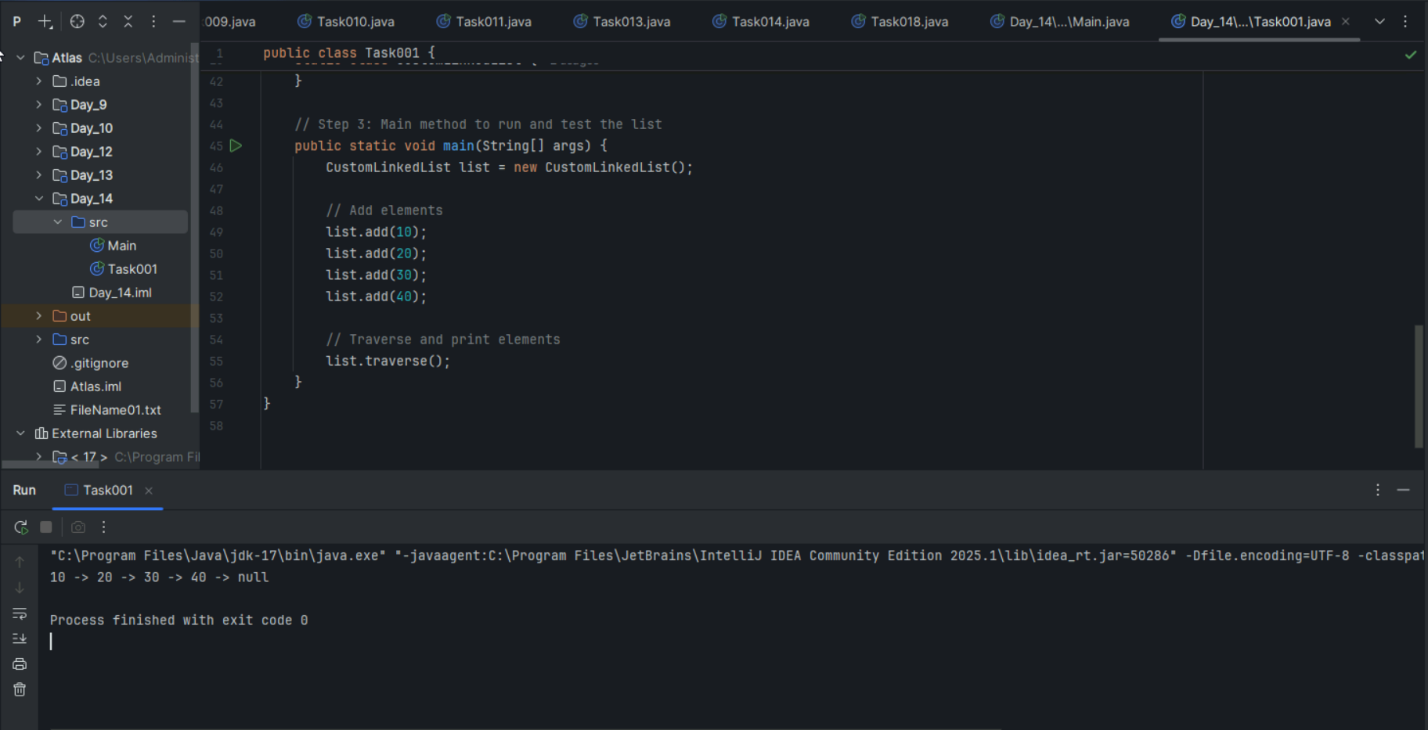
Day 15 – 2nd July 2025

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:** The process of visiting each node in the linked list sequentially, starting from the head node and following the 'next' pointers until reaching the end (null).

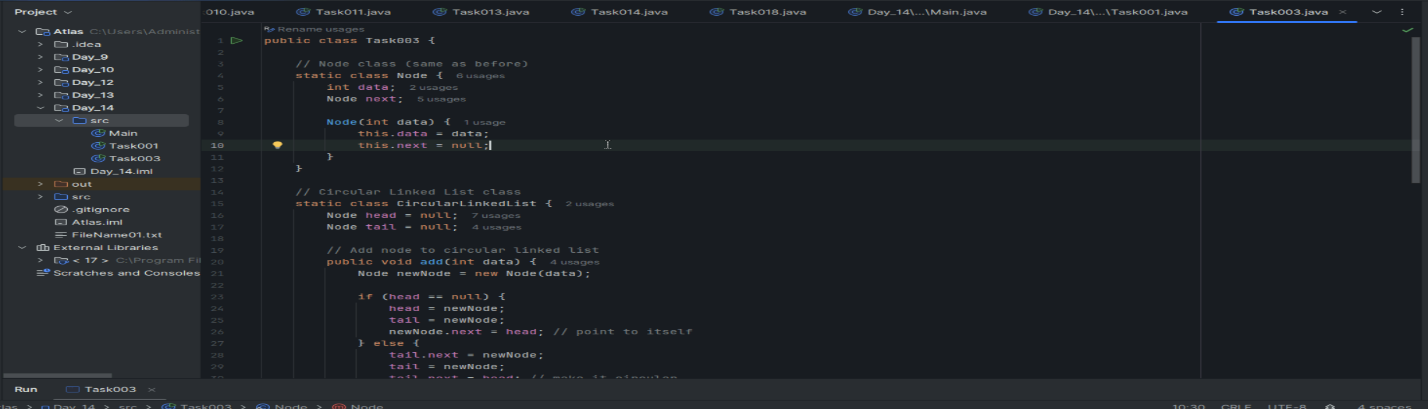
**How it works:** Start at head node → access current node's data → move to next node using 'next' pointer → repeat until 'next' is null.

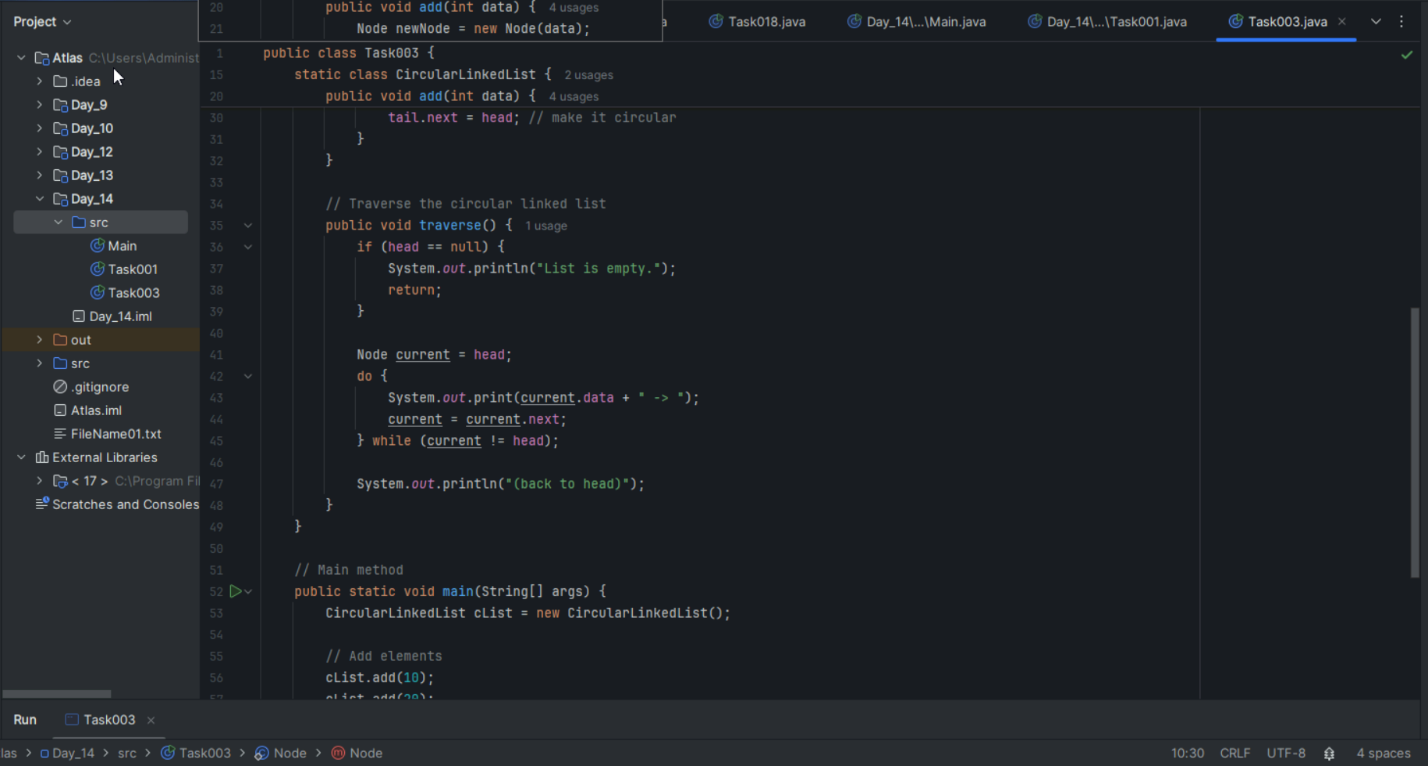
**Purpose:** To read, display, search, or perform operations on all elements in the list. Essential for operations like printing all values, finding specific elements, or calculating sum.

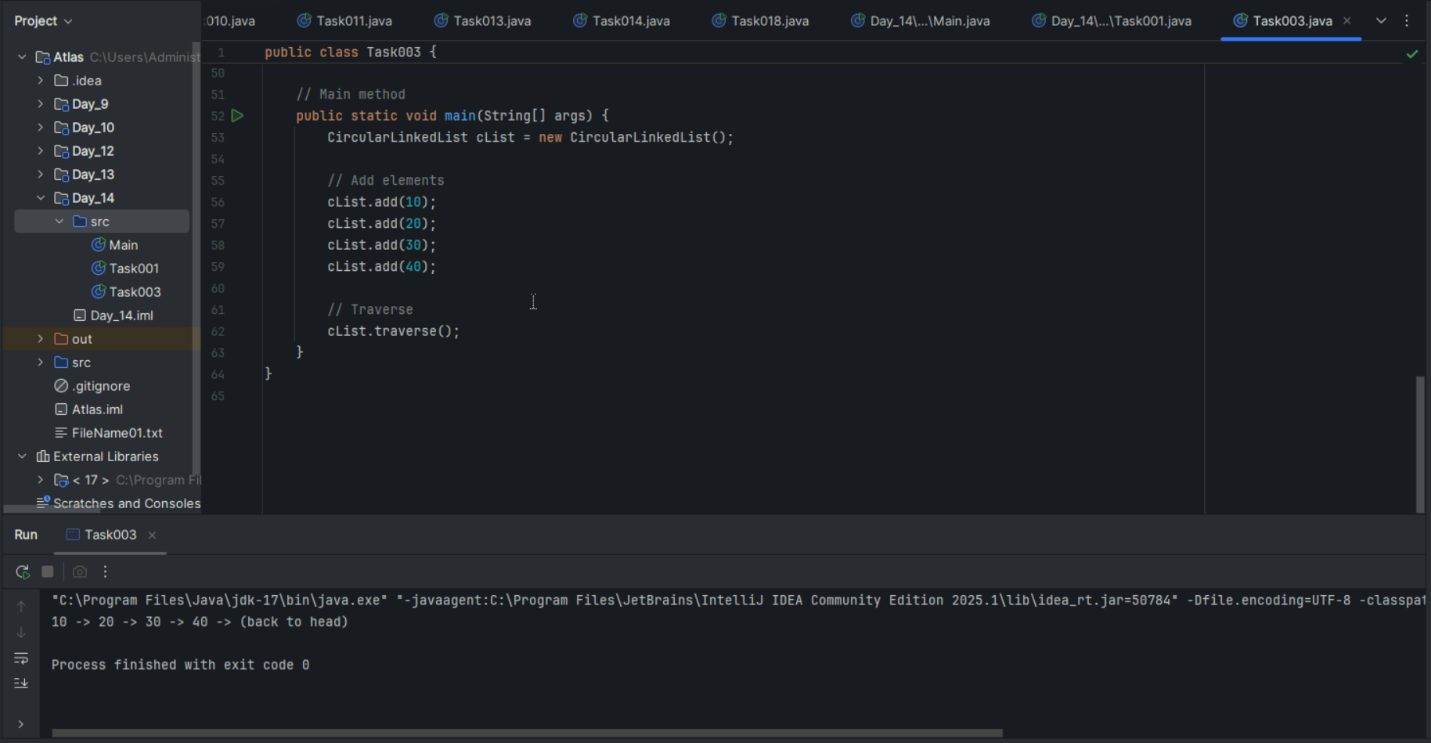
**Time Complexity:** O(n) since we must visit each node once to complete the traversal process.

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");

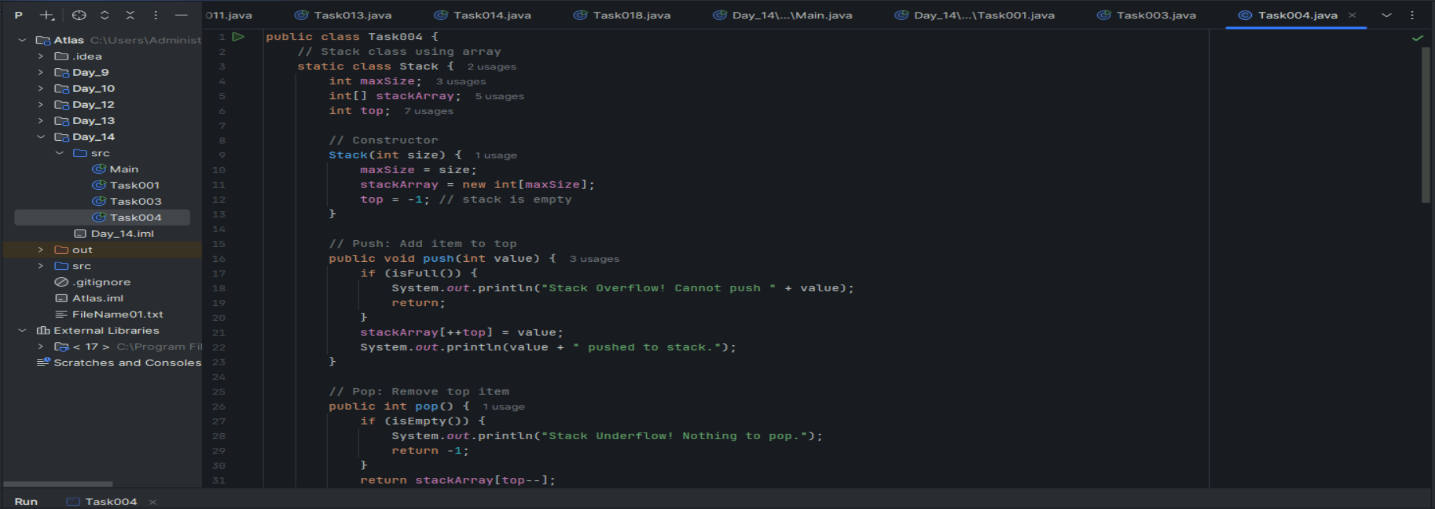
        names.push("Meher");

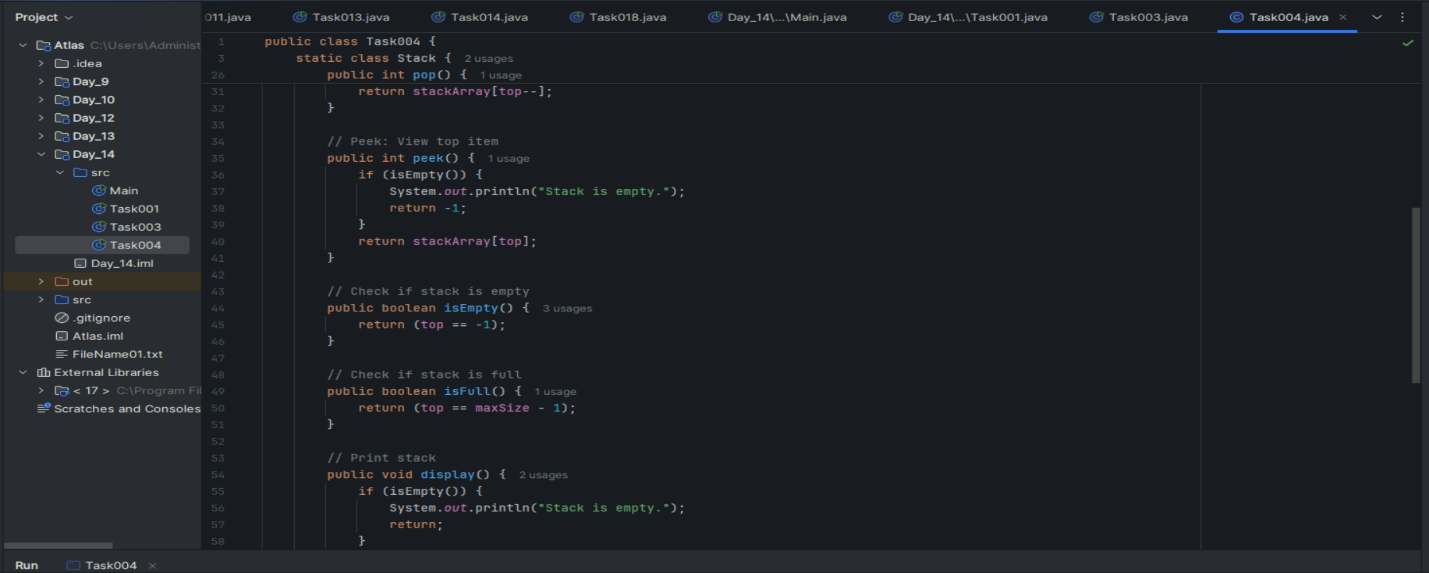
        names.push(".MK");

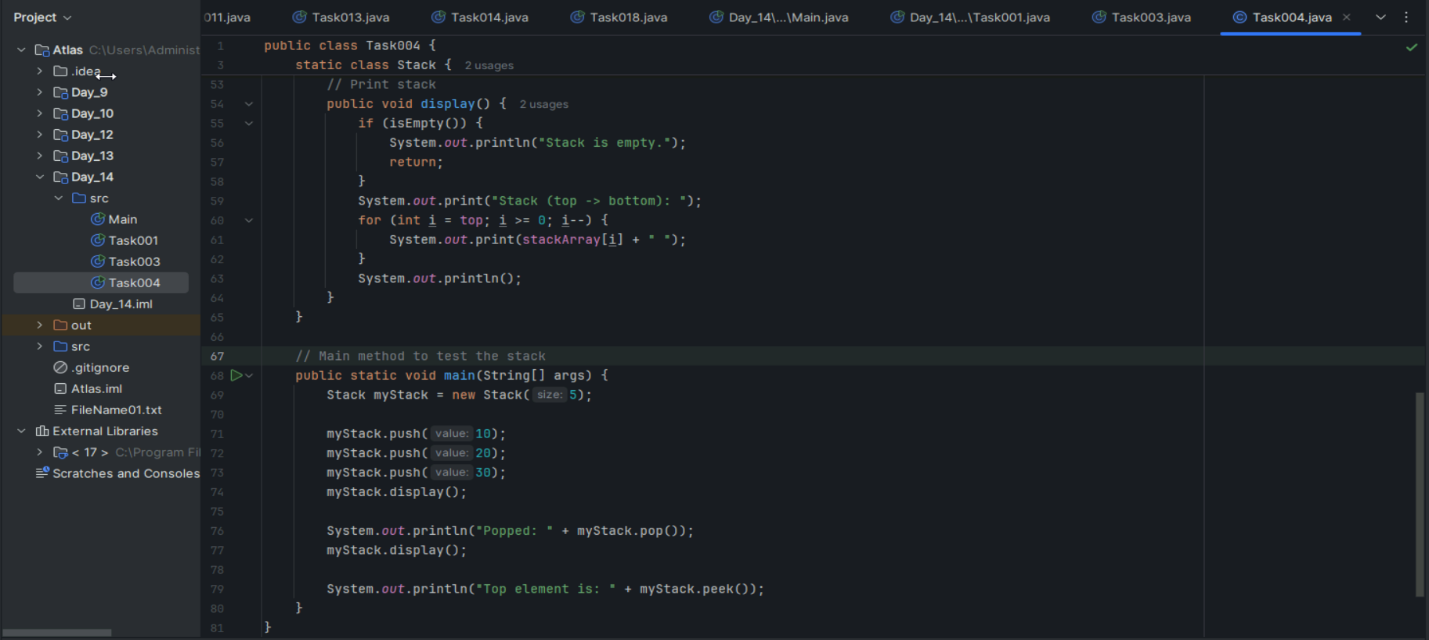
        System.out.println("Stack of names: " + names);

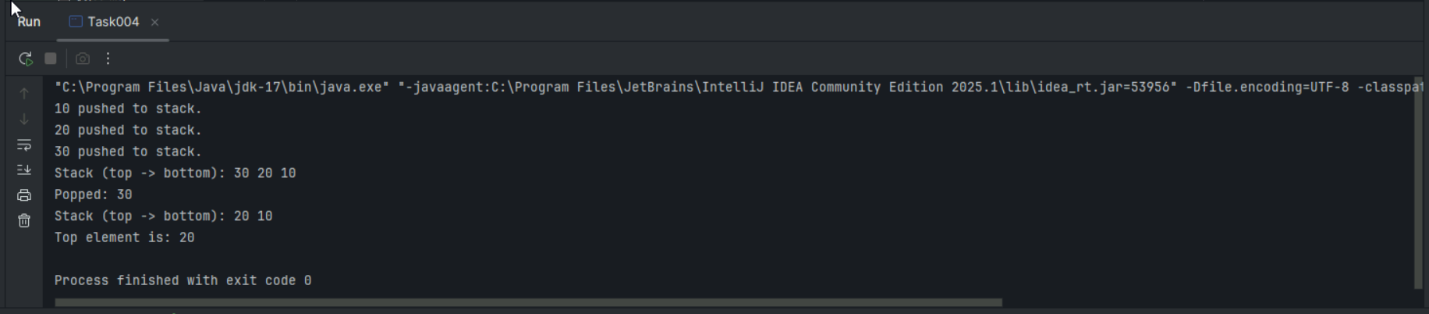
    }

}









Task 5:

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

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 ");

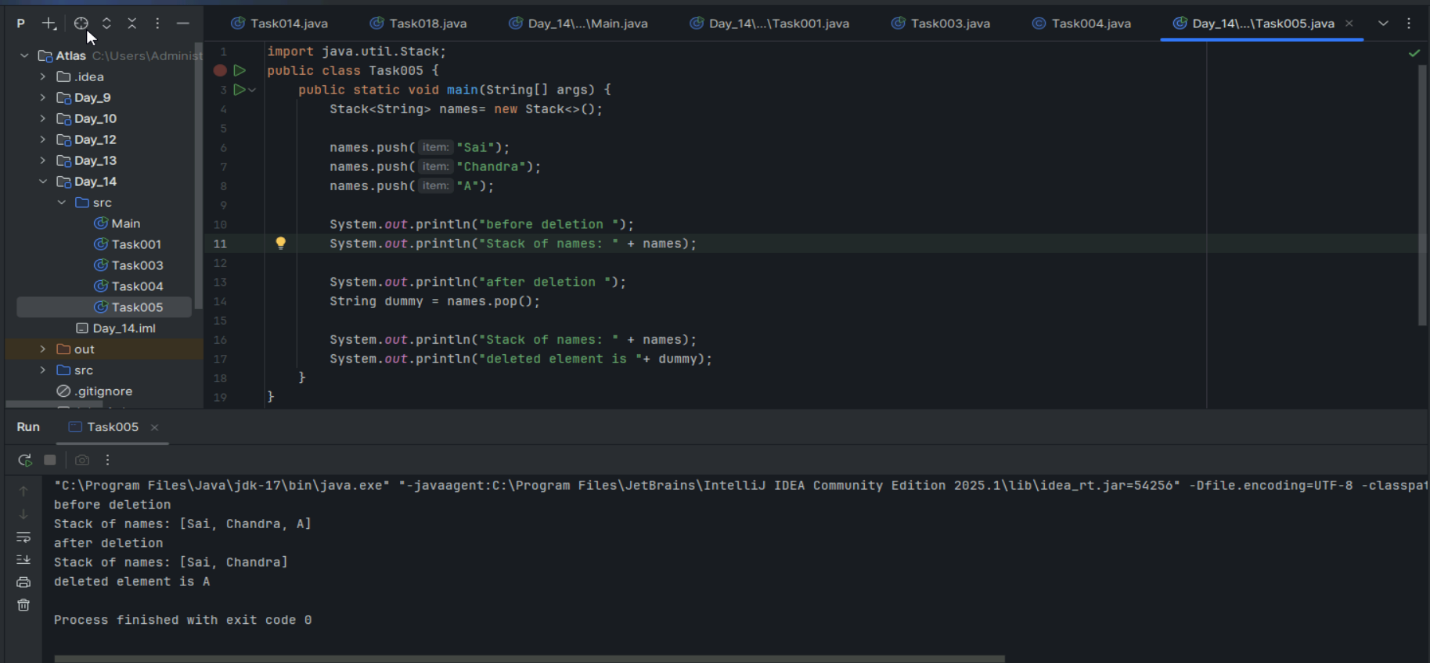
String dummy = names.pop();

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

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

    }

}

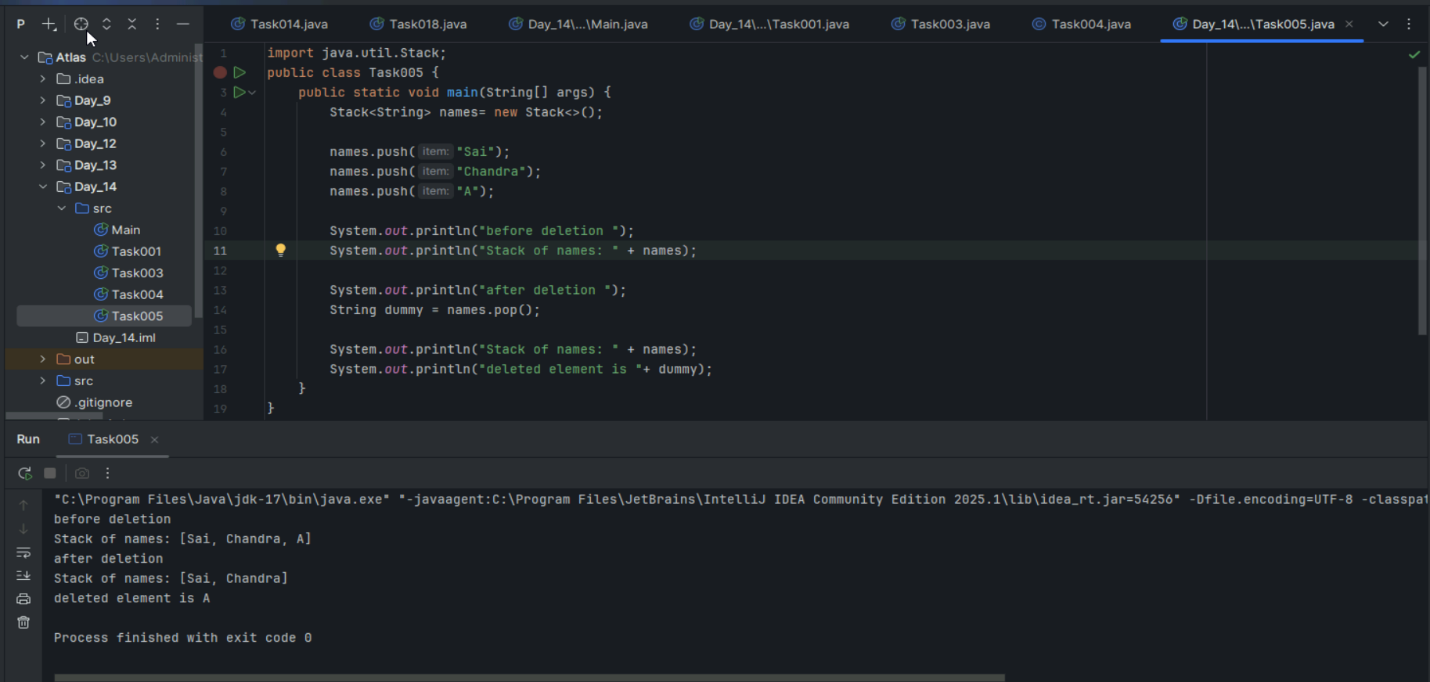


Task 6:

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);

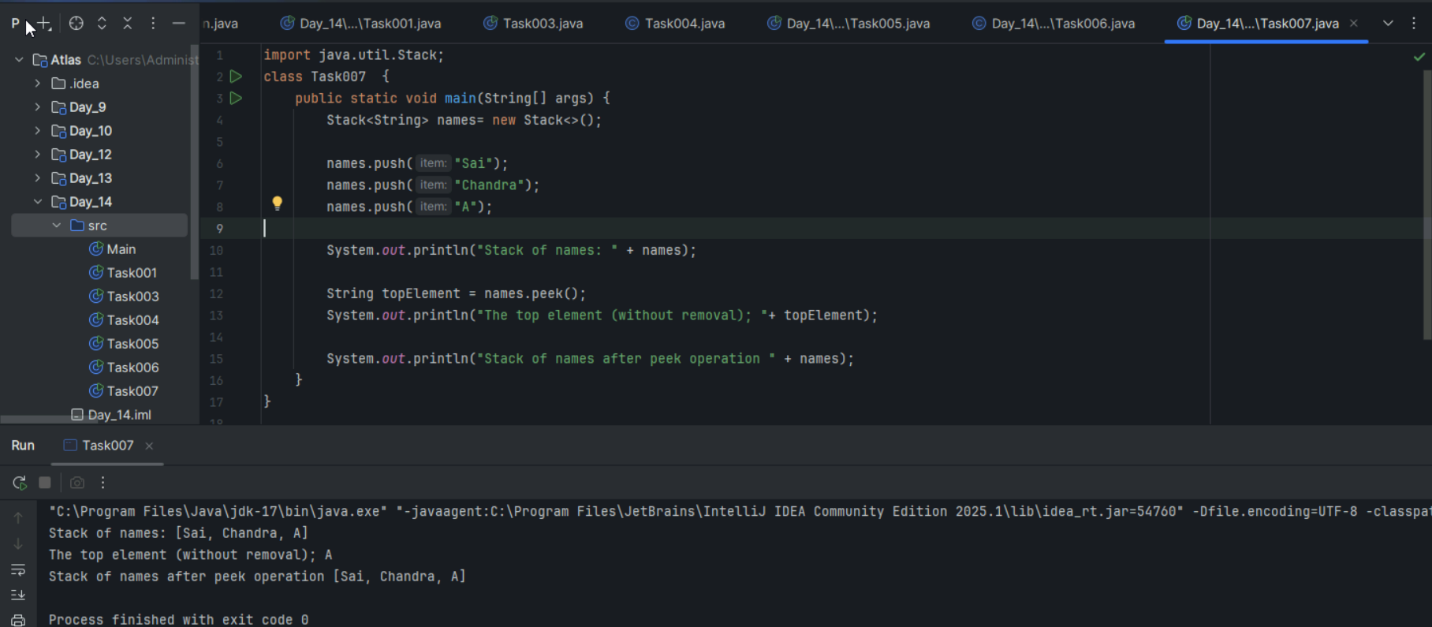
String topElement = names.peek();

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

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

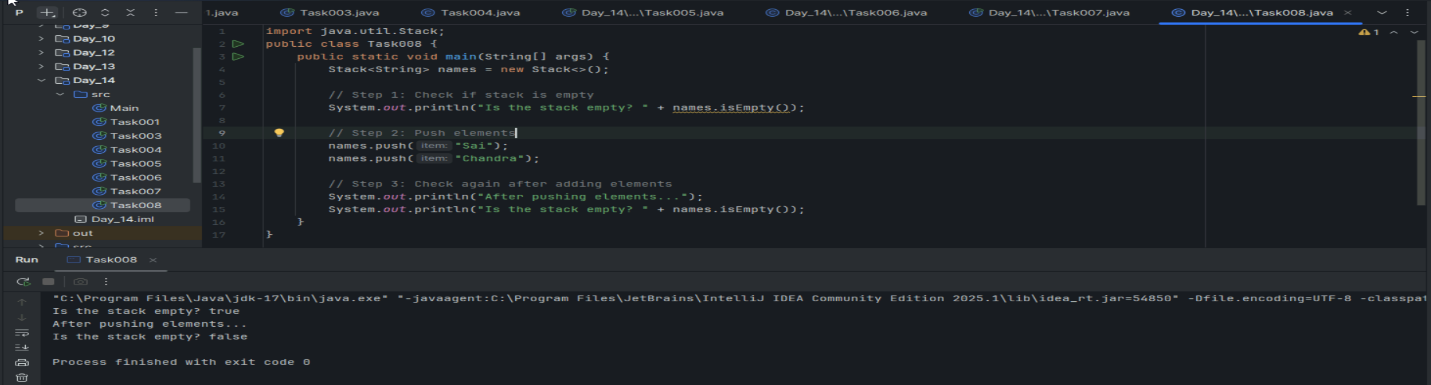
    }

}



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?

**Stack Class Methods:**

**Basic Operations:**

* push(element) - Add element to top of stack
* pop() - Remove and return top element
* peek() / top() - View top element without removing

**Utility Methods:**

* isEmpty() - Check if stack is empty
* size() - Get number of elements in stack
* search(element) - Find position of element from top (1-based)

**Java-specific:** Stack extends Vector, so it also inherits methods like add(), remove(), contains(), but push/pop are preferred for stack operations.

**Key Principle:** LIFO (Last In, First Out) - elements added last are removed first.

Task 10:

What are the common operations in Queues?

**Common Queue Operations:**

**Basic Operations:**

* enqueue() / offer() - Add element to rear/back of queue
* dequeue() / poll() - Remove and return element from front of queue
* front() / peek() - View front element without removing it

**Utility Operations:**

* isEmpty() - Check if queue is empty
* isFull() - Check if queue is full (for bounded queues)
* size() - Get number of elements in queue

**Key Principle:** FIFO (First In, First Out) - elements added first are removed first, like people standing in a line.

**Real-world analogy:** Similar to waiting in line at a store - first person in line gets served first.

Task 11:

Wap to create  a queue with custom methods

Is empty ()

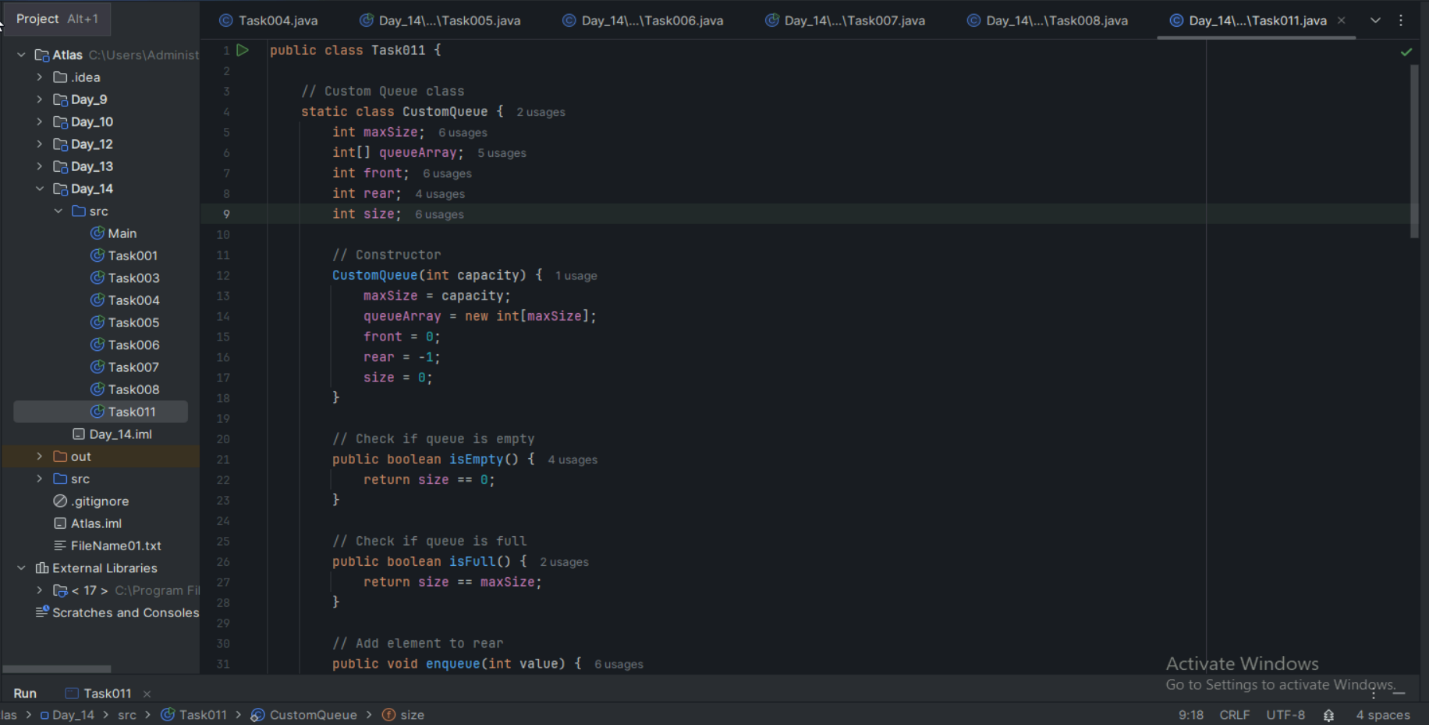
Is full()

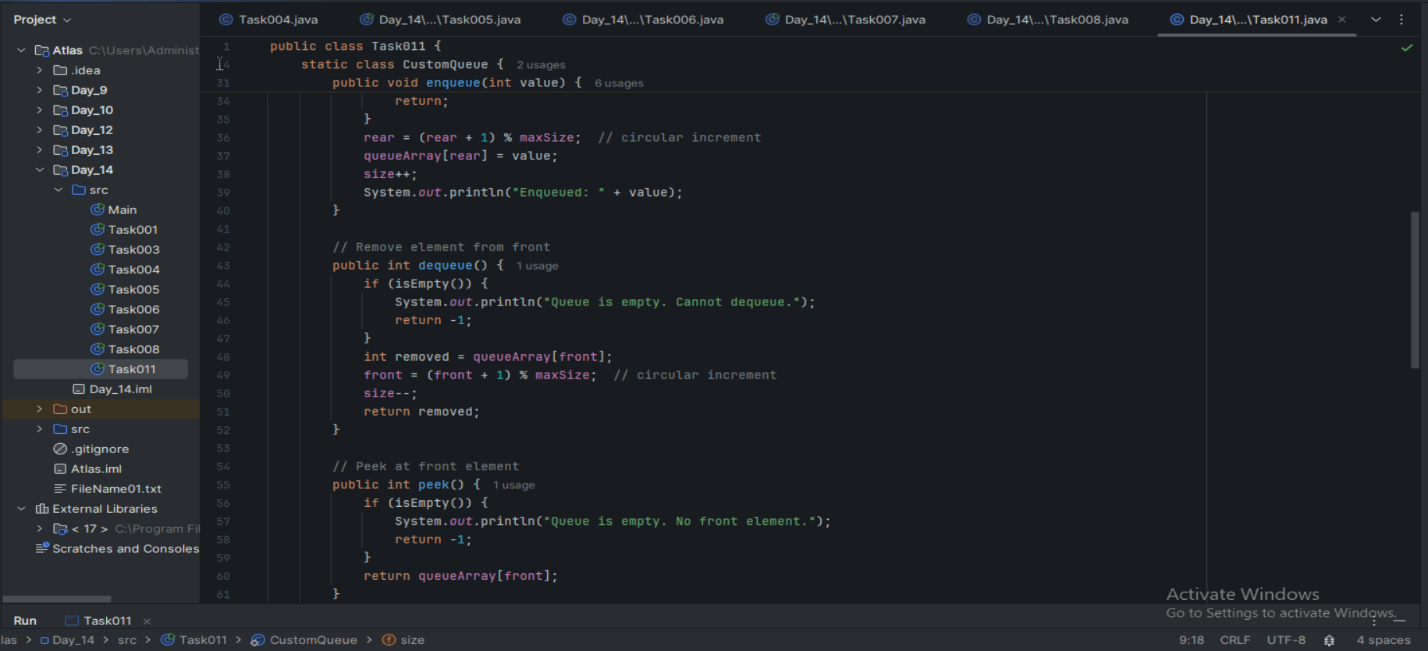
Enque

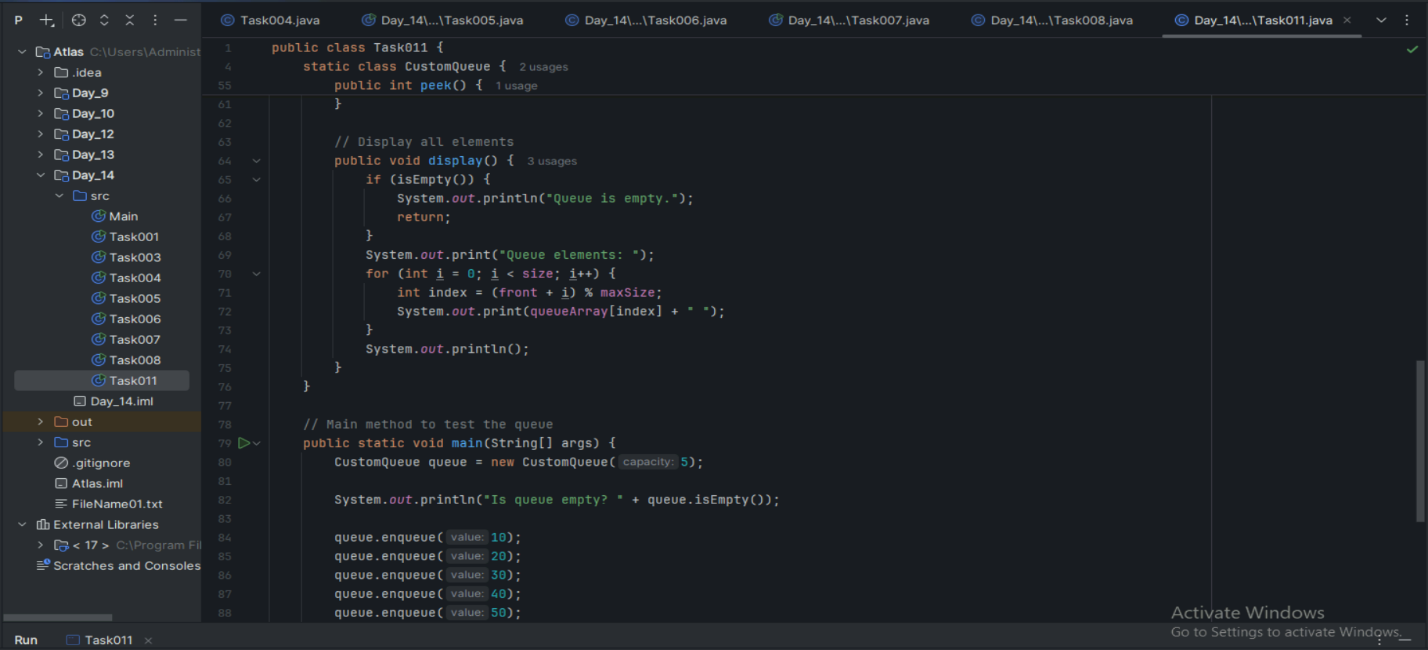
Deque

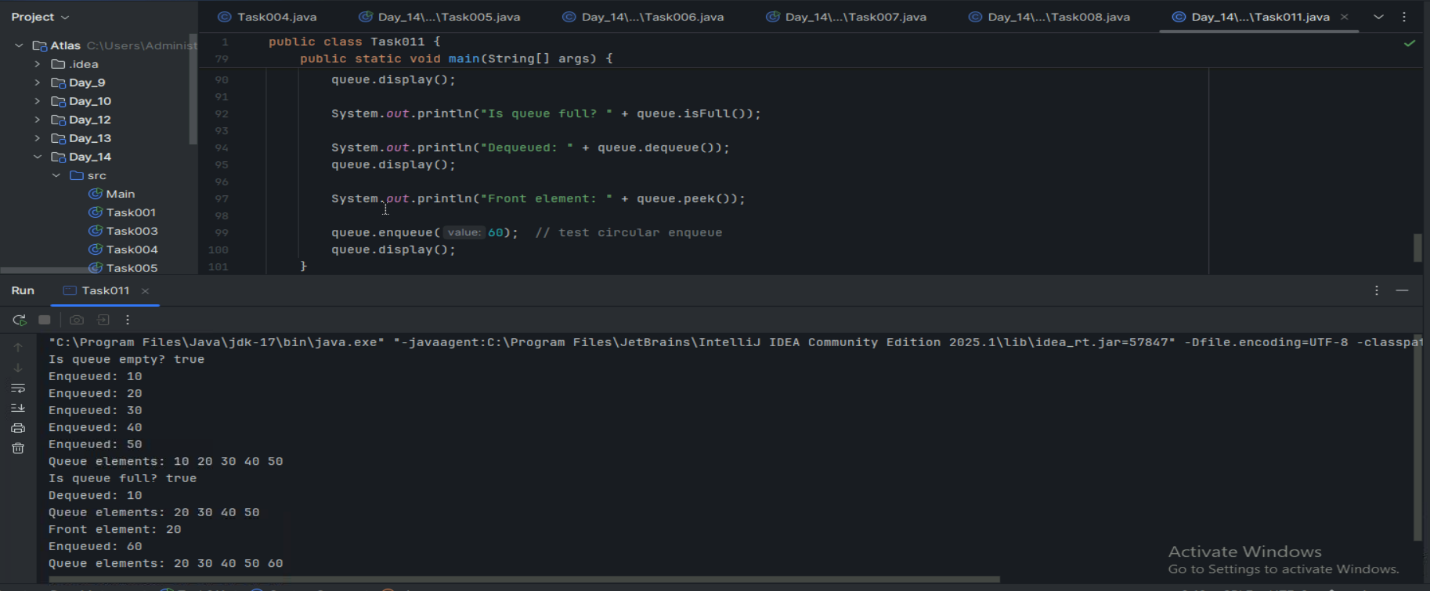
Peek

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();

    }

}

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

Home Tasks:

 Recursion:

Wap to find the factorial of a number

Wap to find the Fibonacci series of a number

What is the difference between recursion and iteration?

Carry forward examples till 10th July

1. Write a recursive function to search for an element in an array

2. Write a recursive function to count the digits of a positive integer (do also for sum of digits)

3. Write a recursive function to reverse a null-terminated string

4. Write a recursive function to convert a decimal number to binary

5. Write a recursive function to check if a string is a palindrome or not

6. Write a recursive function to copy one array to another