Day 14 - 26th June 2025

Linked Lists:

Linked list in c++

#include <bits/stdc++.h>

using namespace std;

// Define a Node class

class Node{

  public:

    int data; // Data part of the node

    Node\* next; // Pointer to the next node

    // Constructor for convenience

    Node(int value) : data(value), next(nullptr) {}

};

// Class for singly linked list

class Linkedlist{

private:

  Node\* head; // Pointer to the head of the list

public:

  // Constructor to initialize an empty list

  Linkedlist(){

    head = nullptr;

  }

  // Function to insert a node at the end

  void insertAtEnd(int value){

    Node\* newNode = new Node(value);

    if(head == nullptr){

      head = newNode; // If list is empty, make newNode the head

    }

    else{

      Node\* temp = head;

      while (temp->next != nullptr){

        temp = temp->next; // Traverse to the last node

      }

      temp->next = newNode; // Link the last node to newNode

    }

  }

  // Function to delete a Node by Value

  void deleteByValue(int value){

    if(head == nullptr){

      return;

    }

    if(head->data == value){

      Node\* temp = head;

      head = head->next; // Move head to the next node

      delete temp; // Free memory of the deleted node

      return;

    }

    Node\* temp = head;

    while(temp->next && temp->next->data != value){

      temp = temp->next; // Traverse to find the node to delete

    }

    if(temp->next){

      Node\* nodeToDelete = temp->next;

      temp->next = temp->next->next; // Unlink the node

      delete nodeToDelete; //Free Memory

    }

  }

  // Function to display the list

  void display(){

    Node\* temp = head;

    while(temp != nullptr){

      cout << temp->data << "->";

      temp = temp->next;

    }

    cout << "NULL" <<endl;

  }

  // Destructor to free all allocated memory

    ~LinkedList() {

        Node\* temp;

        while (head) {

            temp = head;

            head = head->next;

            delete temp;

        }

    }

};

int main() {

    LinkedList list;

    list.insertAtEnd(10);

    list.insertAtEnd(20);

    list.insertAtEnd(30);

    cout << "Linked List: ";

    list.display();

    list.deleteByValue(20);

    cout << "After Deleting 20: ";

    list.display();

    return 0;

}

**Task 001**:

import java.util.LinkedList;

public class Task001\_DS\_LinkedList {

    public static void main(String[] args) {

        LinkedList<String> fruits = new LinkedList<>();

        fruits.add("Apple");

        fruits.add("Banana");

        fruits.addFirst("Orange");

        fruits.addLast("Grapes");

        System.out.println("First Element: " + fruits.getFirst());

        System.out.println("Last Element: " + fruits.getLast());

        fruits.removeFirst();

        fruits.removeLast();

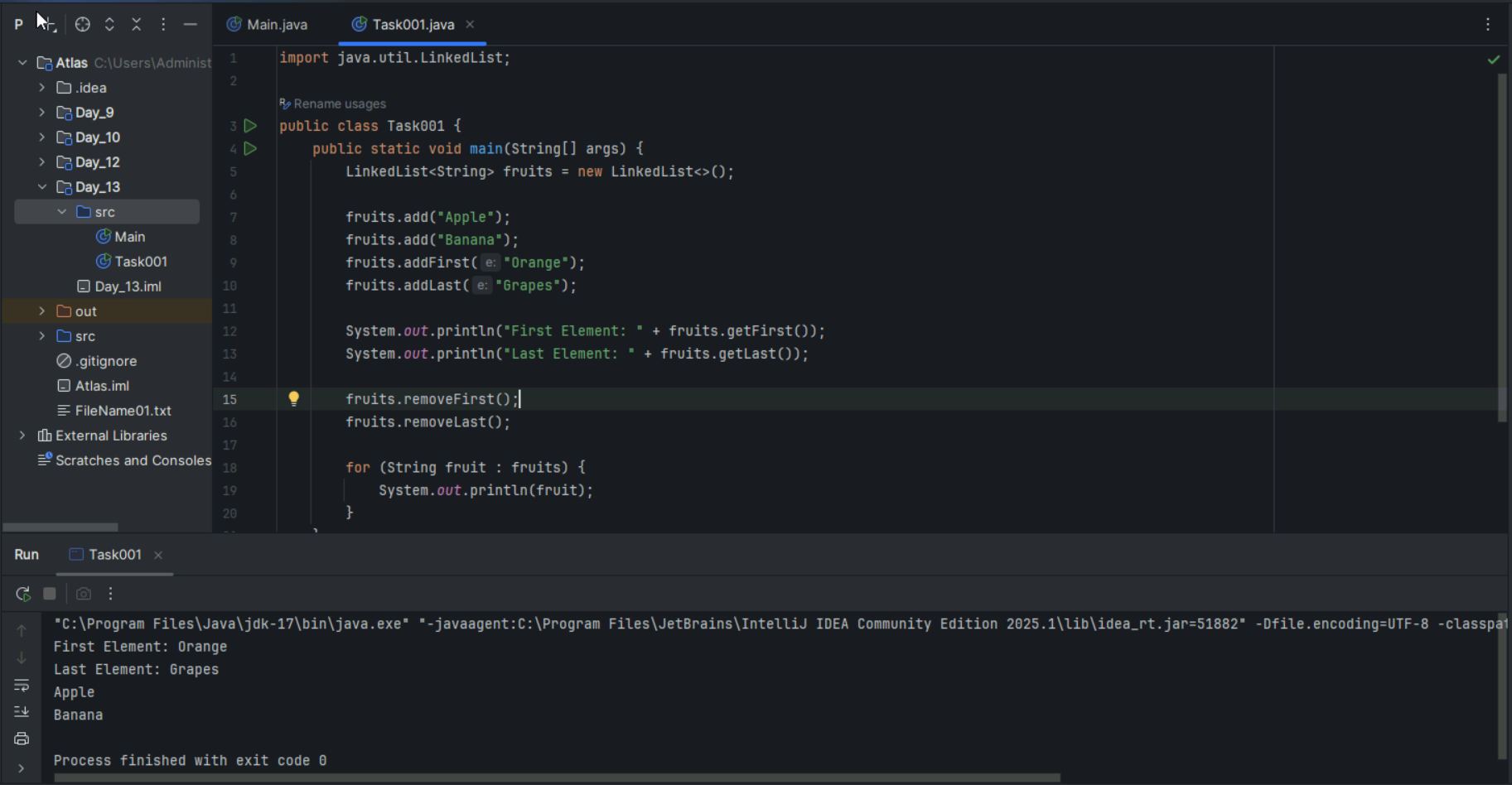
        for (String fruit : fruits) {

            System.out.println(fruit);

        }

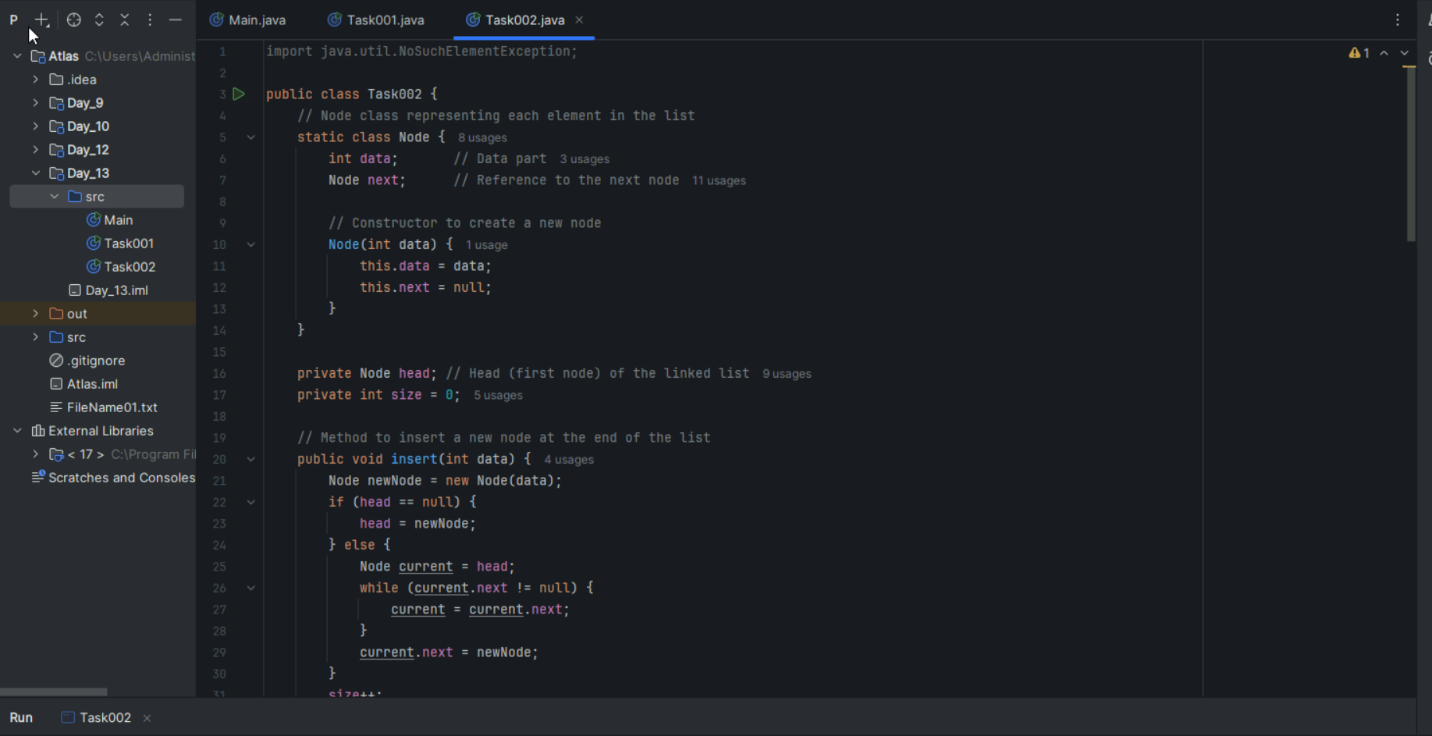
    }

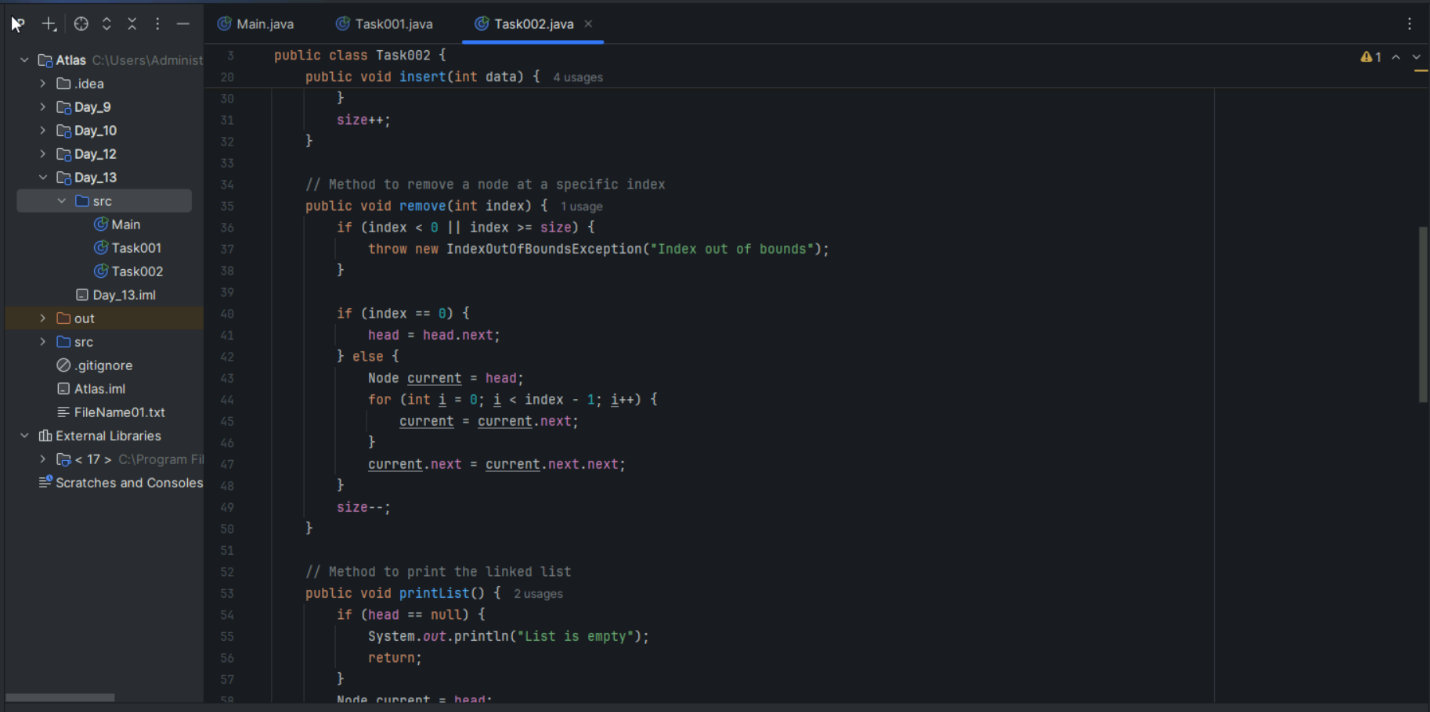
}

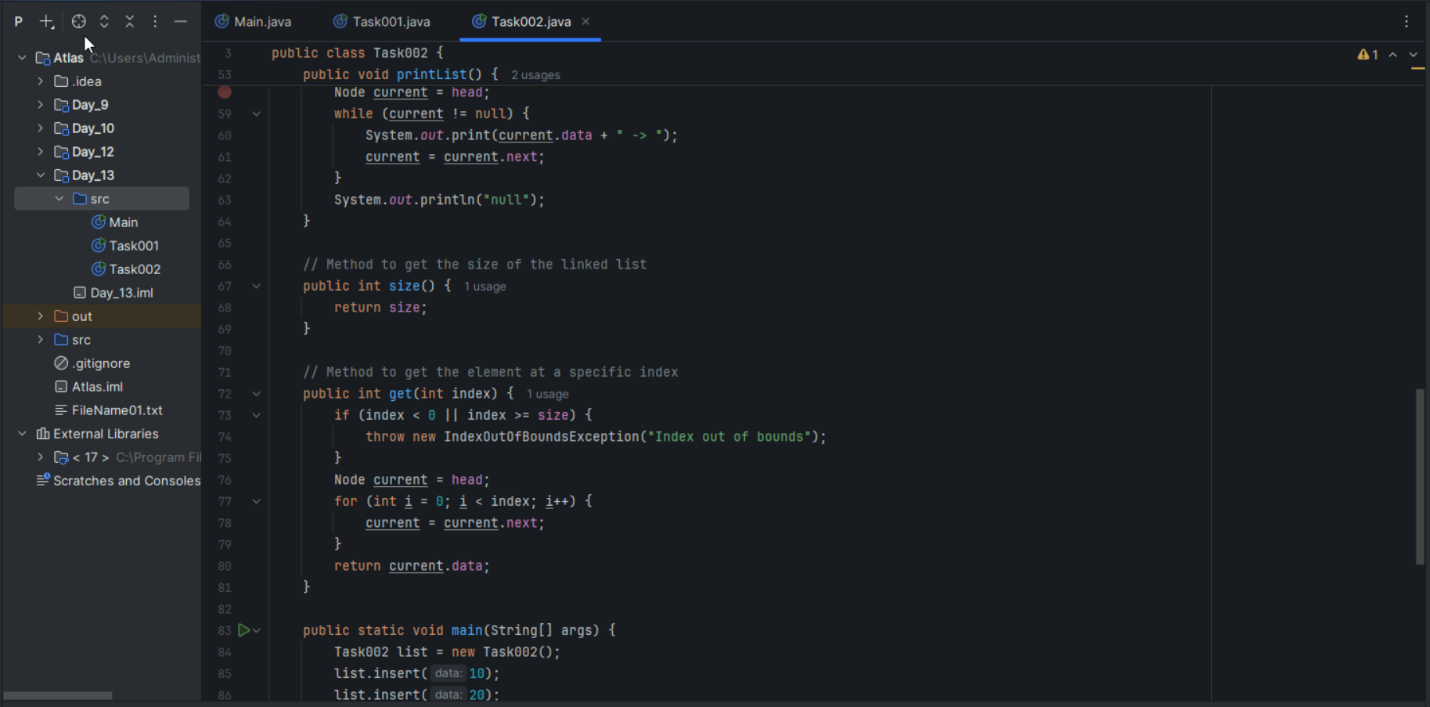


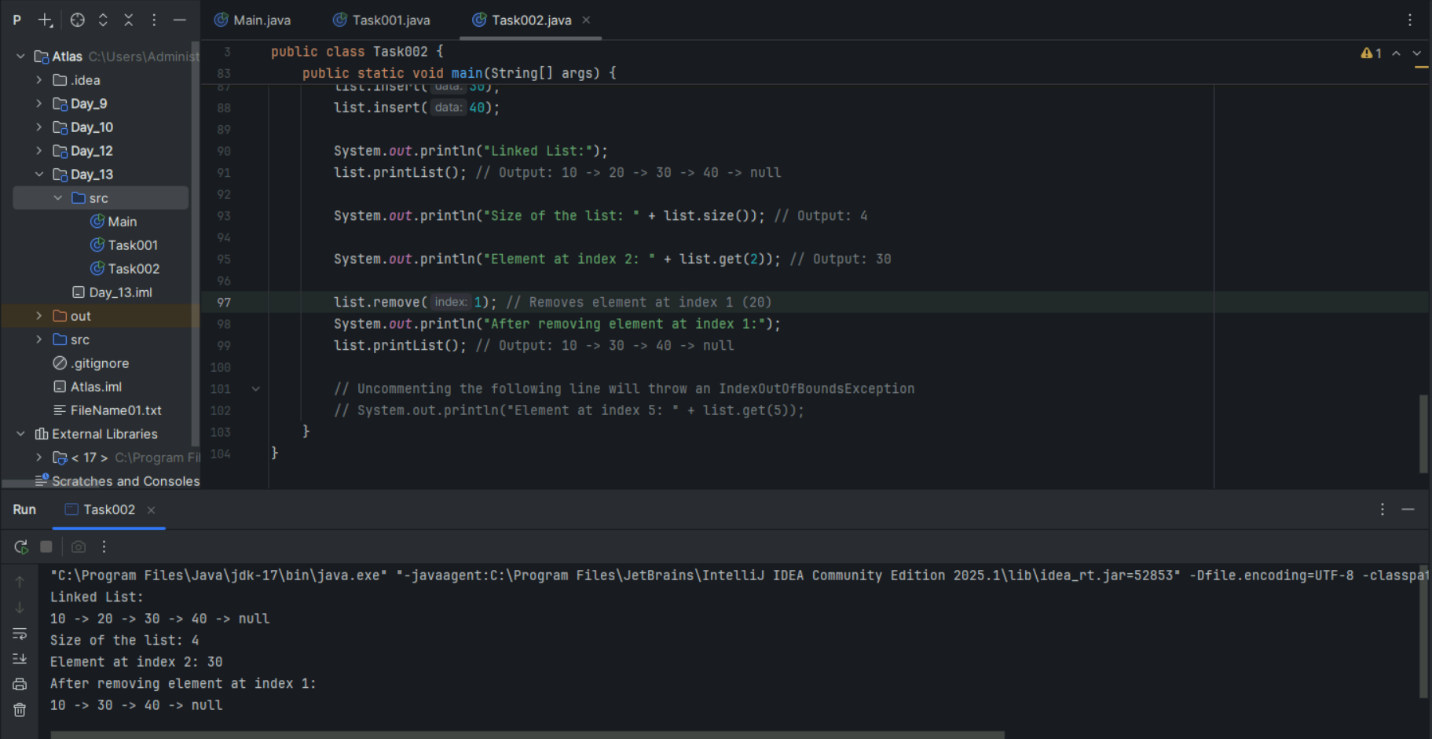
Task 002

Try to create a node and add a value to it..









Task 3

List down the methods of linked lists.

**Linked List Methods:**

**Insertion Methods:**

* addFirst() / insertAtBeginning() - Add node at start
* addLast() / insertAtEnd() - Add node at end
* insertAtPosition() - Add node at specific index

**Deletion Methods:**

* removeFirst() / deleteFromBeginning() - Remove first node
* removeLast() / deleteFromEnd() - Remove last node
* remove(value) / deleteByValue() - Remove node with specific value

**Search & Access:**

* search(value) - Find if value exists
* get(index) - Retrieve value at specific position
* indexOf(value) - Get index of specific value

**Utility Methods:**

* size() - Get total number of nodes
* isEmpty() - Check if list is empty
* display() / print() - Show all elements

**Task 4:**

What are the operations of data structures? I liner

**Data Structure Operations (One-liner):**

**Insertion** - Adding elements, **Deletion** - Removing elements, **Traversal** - Visiting all elements, **Search** - Finding specific elements, **Update** - Modifying existing elements, **Sort** - Arranging elements in order.

Traversing

Insertion

Deletion

Searching

Sorting

**Task 5**:

Wap to create a linked list add 5 elements to it and replace 3rd element with different value.

Hint:

import java.util.\*;

public class Task005\_DS\_Linkedlist {

    public static void main(String args[]) {

        LinkedList<String> ll = new LinkedList<>();

        ll.add("Prasunamba");

        ll.add("Meher");

        ll.add(1, ".Mk");

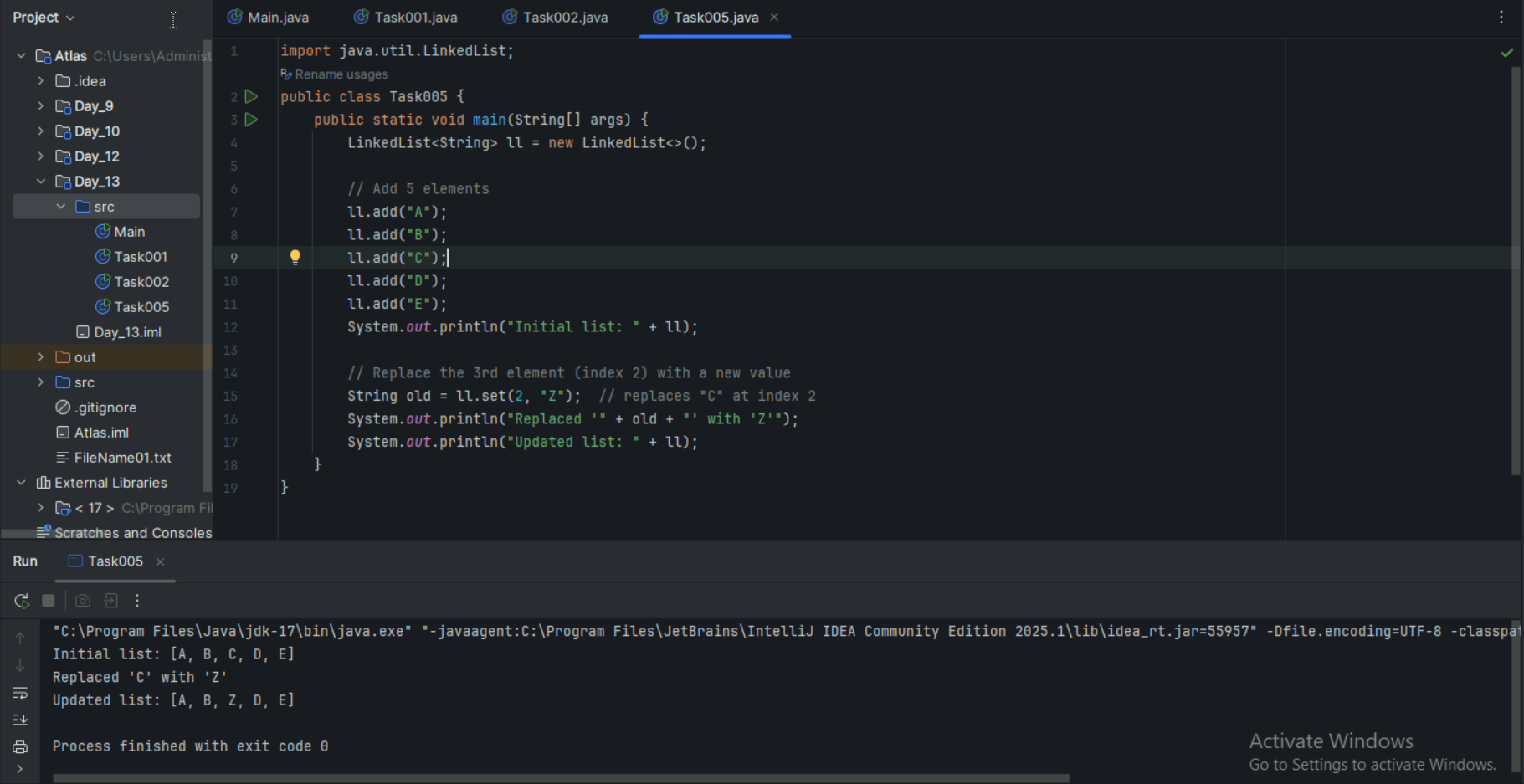
        System.out.println("Initial LinkedList " + ll);

        ll.set(1, "MP");

        System.out.println("Updated LinkedList " + ll);

    }

}



**Task 6**:

Wap to create a linked list to add 5 elements and remove any element and display

14.56 to 15.00

Hint:

// Java program to remove elements

// in a LinkedList

import java.util.\*;

public class Task006\_DS\_Linkedlist {

    public static void main(String args[])   {

        LinkedList<String> ll = new LinkedList<>();

        ll.add("Prasunamba");

        ll.add("Meher");

        ll.add(1, ".MK");

        System.out.println("display LinkedList " + ll);

        ll.remove(1);

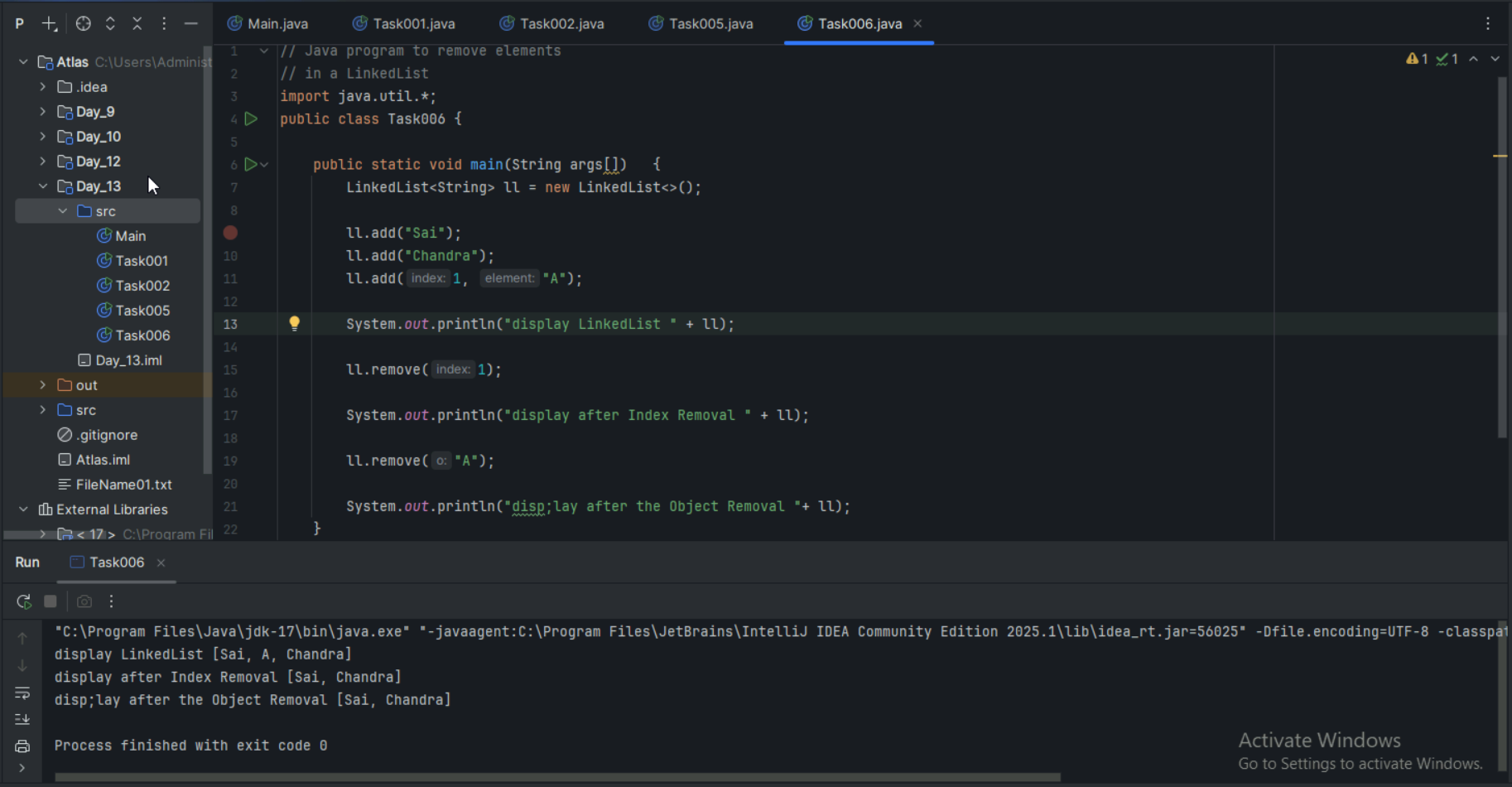
        System.out.println("display after Index Removal " + ll);

        ll.remove(".MK");

        System.out.println("disp;lay after the Object Removal "+ ll);

    }

}



**Task 7**:

Wap to create a linked list to add 5 elements and display the list using for (use get() ) and for each loops

import java.util.\*;

public class Task007\_DS\_Linkedlist {

    public static void main(String args[]) {

        LinkedList<String> ll = new LinkedList<>();

        ll.add("Prasunamba");

        ll.add("Meher");

        ll.add(1, ".MK");

        // we are using get () with for loop

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

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

        }

        System.out.println();

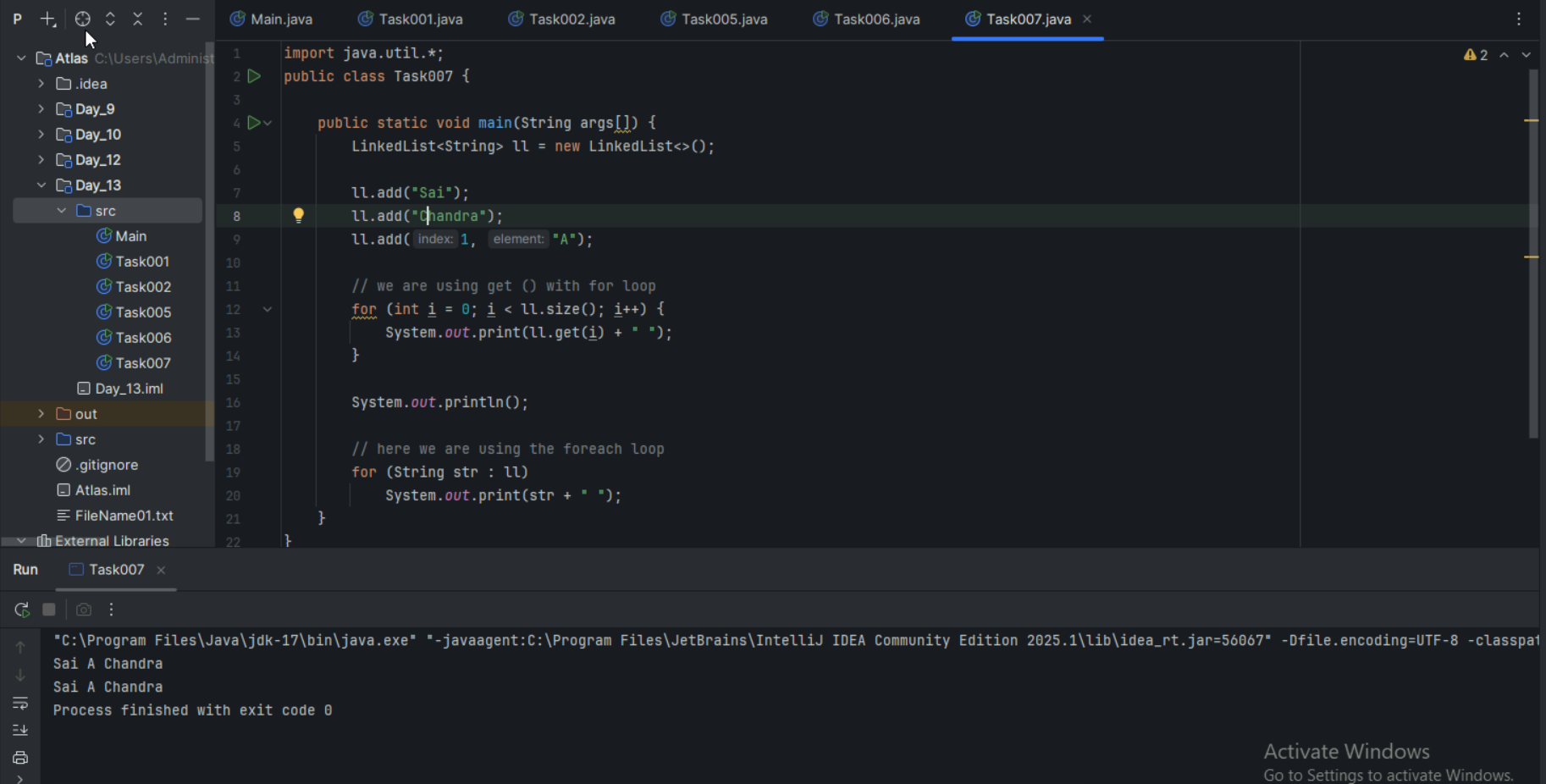
        // here we are using the foreach loop

        for (String str : ll)

            System.out.print(str + " ");

    }

}



**Task 8**:

Createa  a linked list and few items and convert it into an array

154.10 to 15.15

Hint:

import java.util.\*;

public class Task008\_DS\_Linkedlist {

    public static void main(String[] args)   {

        LinkedList<Integer> ll= new LinkedList<Integer>();

        ll.add(5000);

        ll.add(1000);

        ll.add(8000);

        ll.add(7000);

        System.out.println("display : "+ ll);

        Object[] a = ll.toArray();

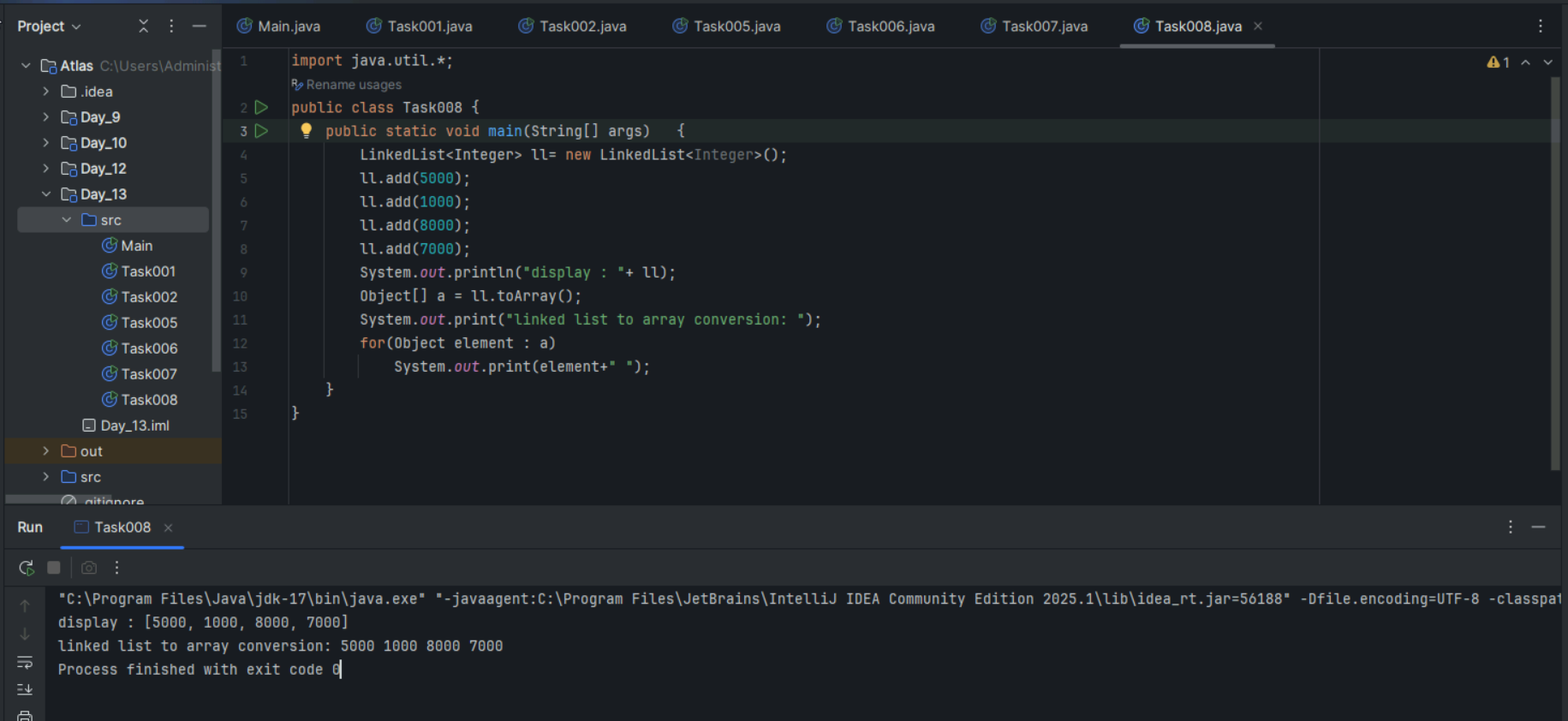
        System.out.print("linked list to array conversion: ");

        for(Object element : a)

         System.out.print(element+" ");

    }

}



**Task 9**:

Create a linked list add few items and clone the 1st linked list with the 2nd linked list

Hint:

import java.util.LinkedList;

public class Task009\_DS\_Linkedlist\_clone {

    public static void main(String args[]) {

        LinkedList<String> ll1 = new LinkedList<>();

LinkedList ll2 = new LinkedList();

        l.add("Prasunamba");

        l.add("Meher");

        l.add(".MK");

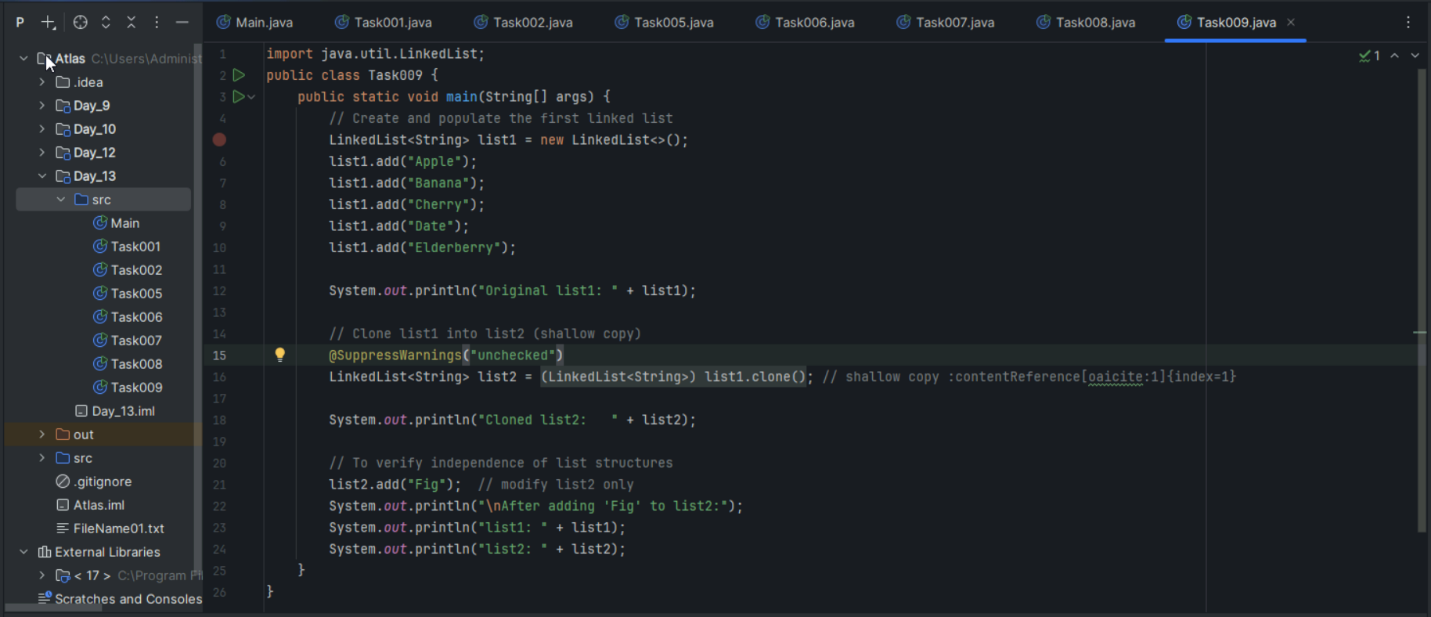
        System.out.println("my original LinkedList contains\n: " + ll1);

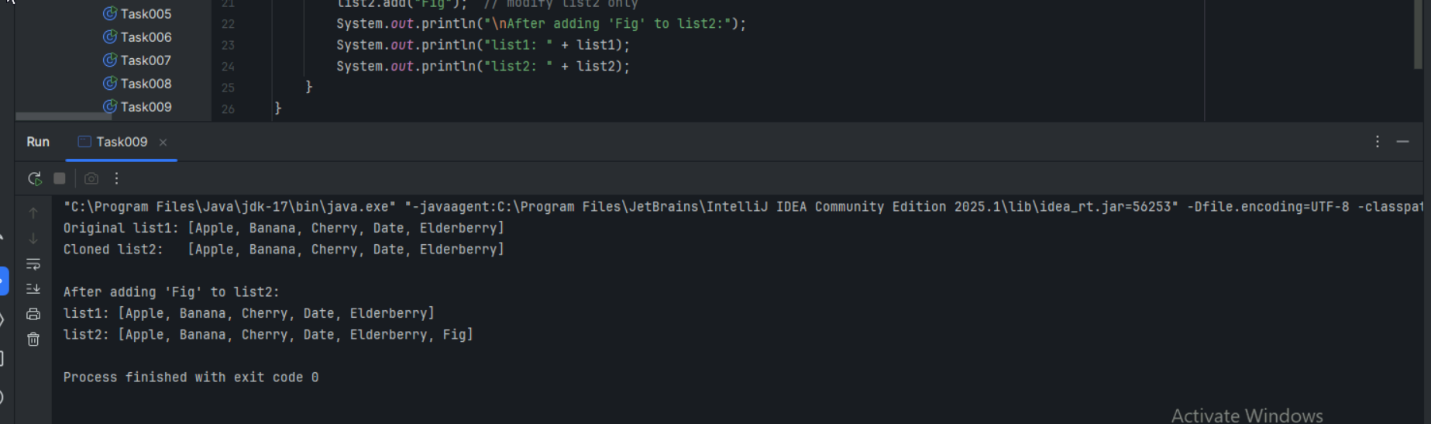
        ll2 = (LinkedList)ll1.clone();

        System.out.println("after cloning ll2 has \n: " + ll2);

    }

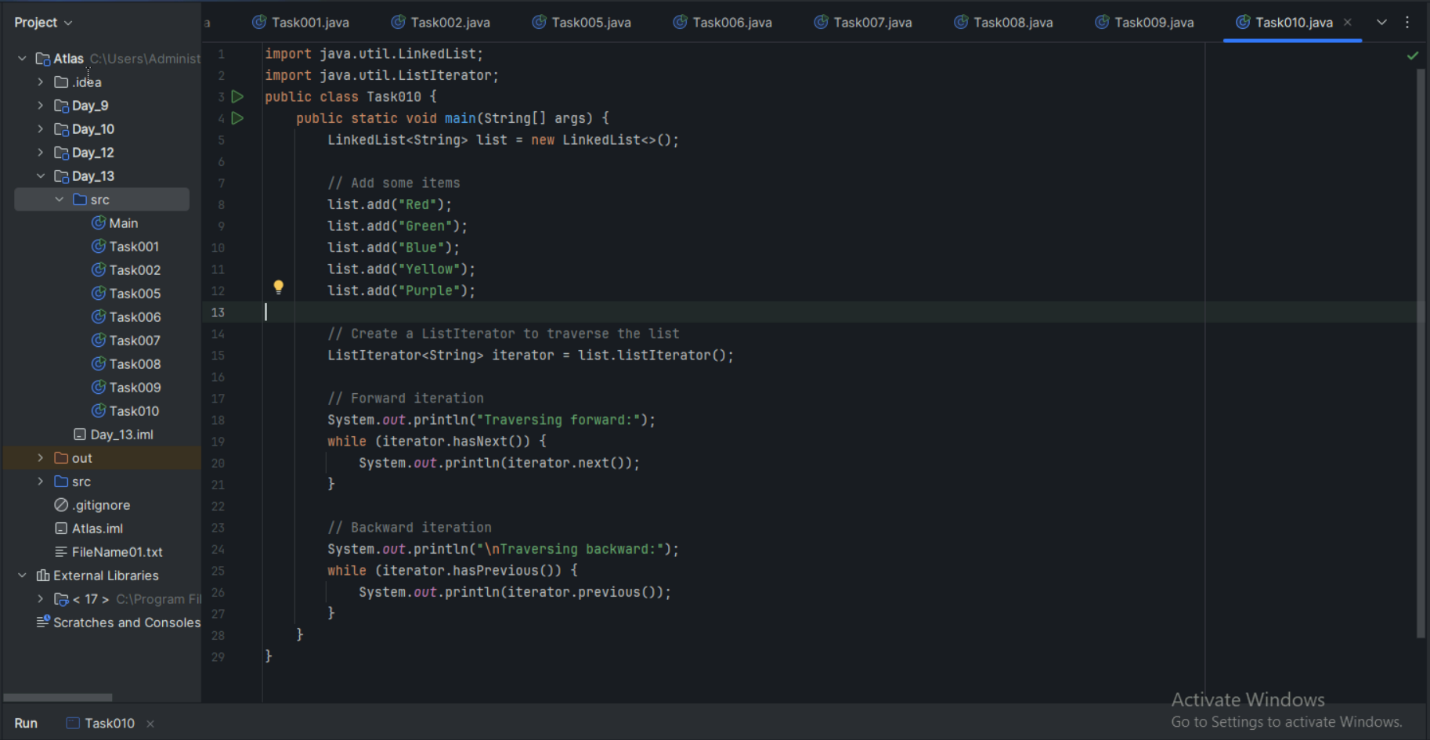
}

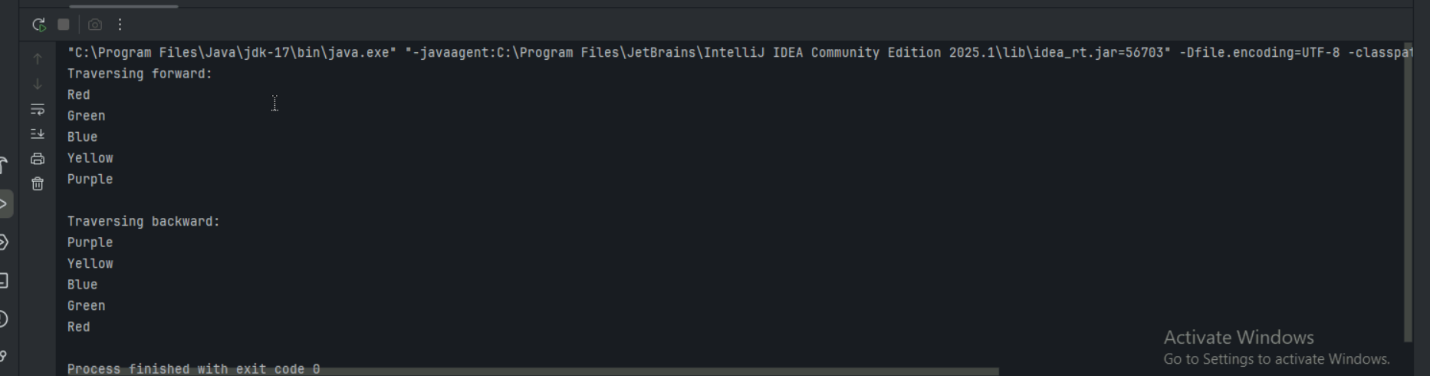




**Task 10**:

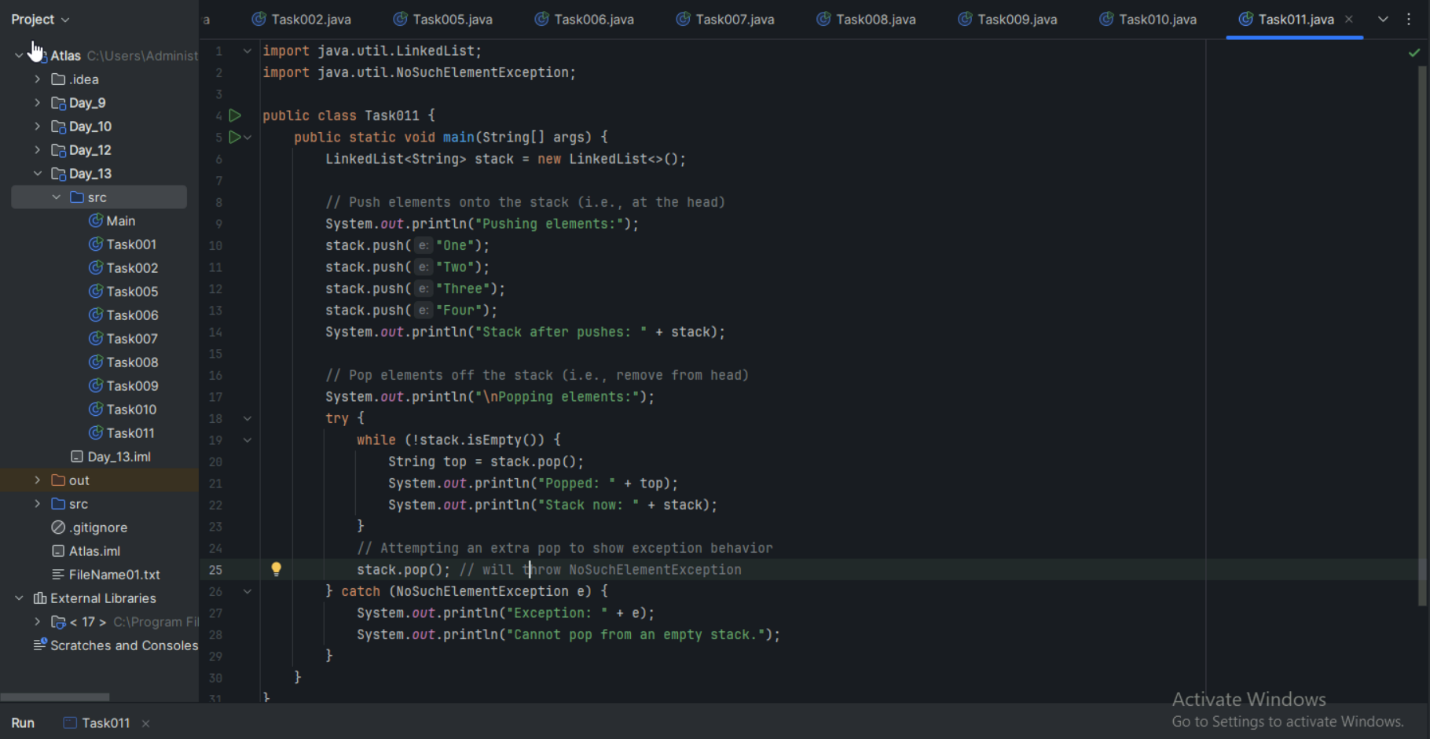
Create  linked list and iterate the values using ListIterator class in util package

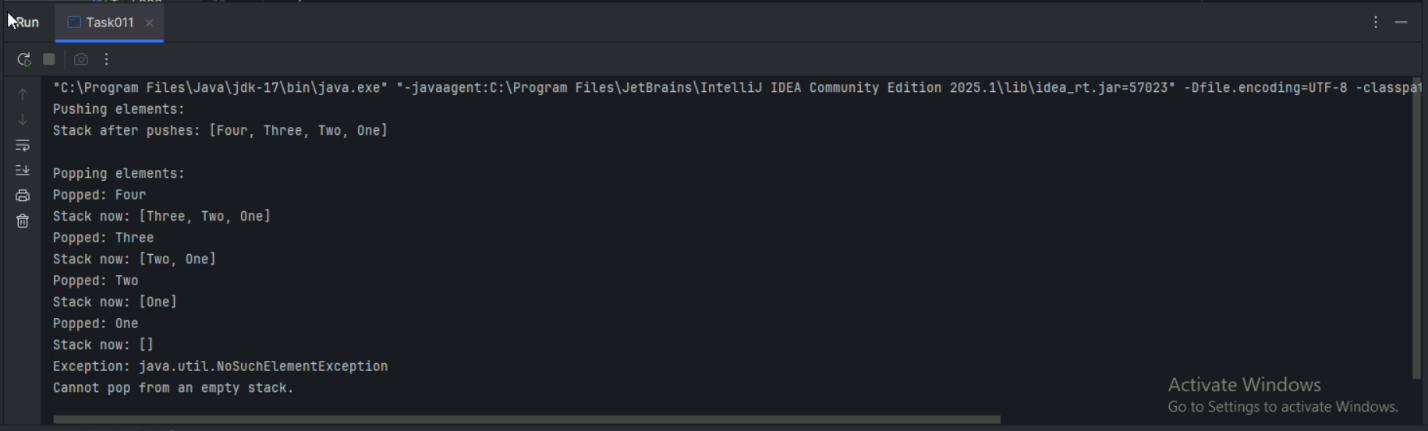




Task 11:

Create a linked list and use push and pop methods.





Task 12:

Difference between Iterator and splitIterator

**Iterator vs Spliterator:**

**Iterator:**

* **Purpose:** Sequential traversal of collections one element at a time
* **Methods:** hasNext(), next(), remove()
* **Parallelism:** Single-threaded, no parallel processing support
* **Usage:** Basic iteration through List, Set, Map collections

**Spliterator:**

* **Purpose:** Parallel traversal and splitting of data sources for concurrent processing
* **Methods:** tryAdvance(), trySplit(), estimateSize(), characteristics()
* **Parallelism:** Supports parallel streams and multi-threaded operations
* **Usage:** Stream API operations, parallel processing with forEach(), filter(), map()

**Key Difference:** Iterator is for sequential processing, Spliterator enables parallel processing by splitting data into chunks for multiple threads.

**Task 13**:

Below is the code for Split iterator… run it to see the output..

Can you try it to out()... and see ..

import java.util.\*;

public class Task0013\_DS\_Linkedlist\_SplitIterator {

    public static void main(String[] args) {

        LinkedList<String> lobj = new LinkedList<>();

        lobj.add("Prasunamba");

        lobj.add("Meher");

        lobj.add(".MK");

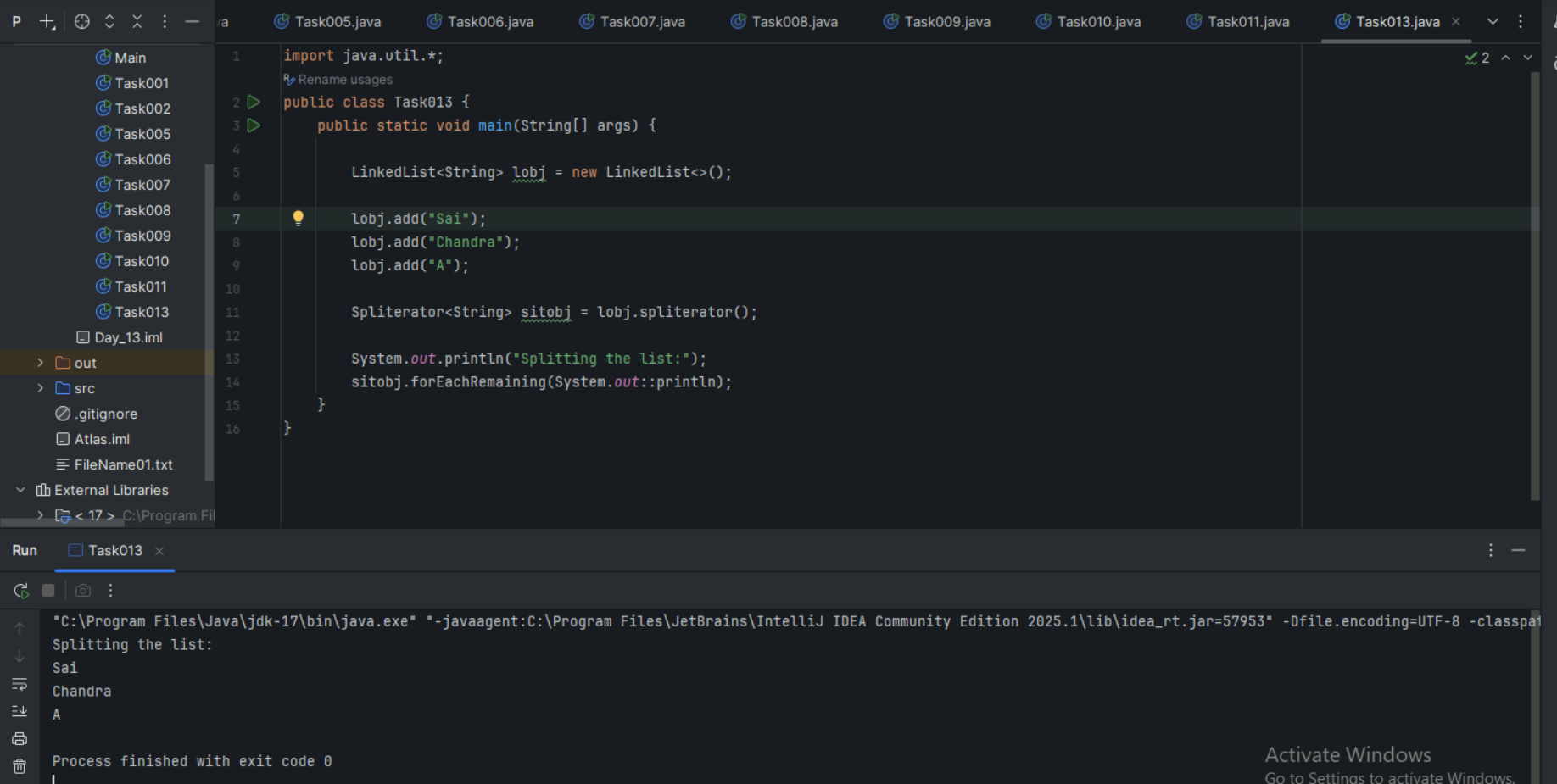
        Spliterator<String> sitobj = lobj.spliterator();

        System.out.println("Splitting the list:");

        sitobj.forEachRemaining(System.out::println);

    }

}



**Task 14**:

Create alinkedlist and display items into 2 lists using split  iterator

Hint:

Spliterator<String> itobj2 = itobj1.trySplit();

while( itobj1.tryAdvance( (n) -> { System.out.println(n); } ) );

import java.util.LinkedList;

import java.util.Spliterator;

public class Task0014\_DS\_Linkedlist\_SplitItr2Lists {

public static void main(String[] args) {

    LinkedList<String> llobj = new LinkedList<String>();

    llobj.add("Prasunamba");

    llobj.add("Meher");

    llobj.add(".MK");

    llobj.add("MP");

    Spliterator<String> itobj1 = llobj.spliterator();

    Spliterator<String> itobj2 = itobj1.trySplit();

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

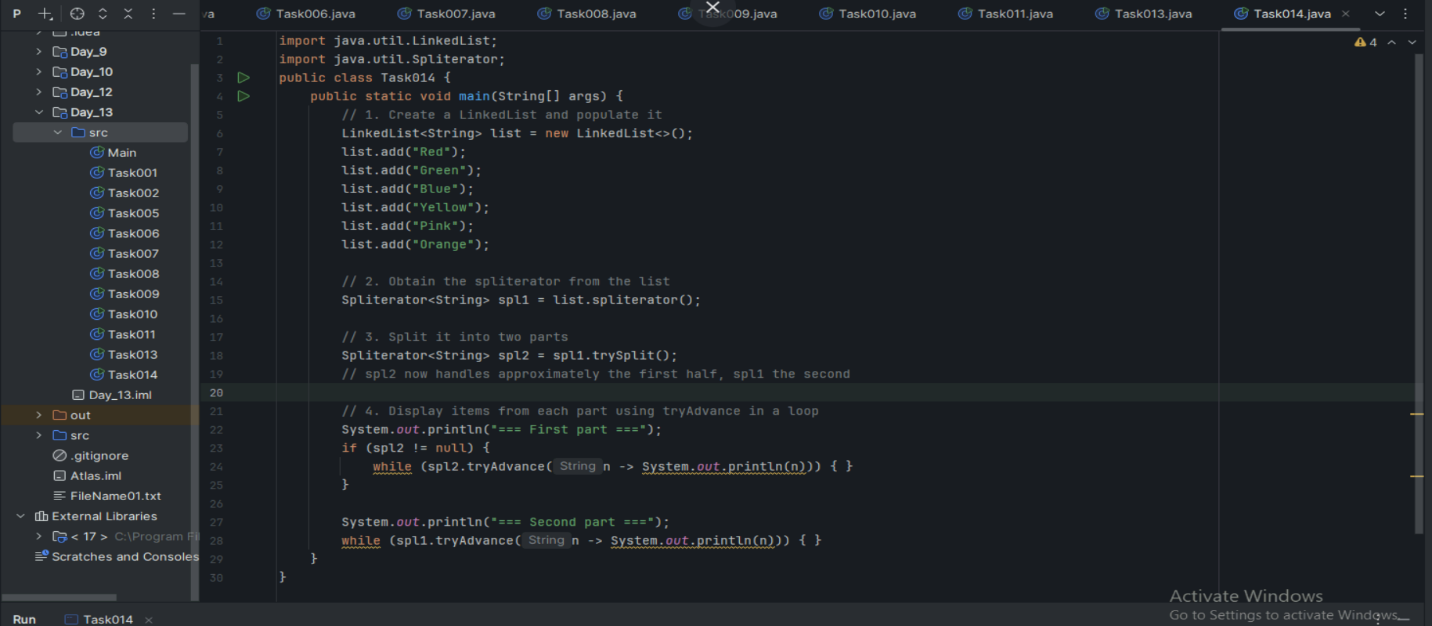
    while( itobj1.tryAdvance( (n) -> { System.out.println(n); } ) );

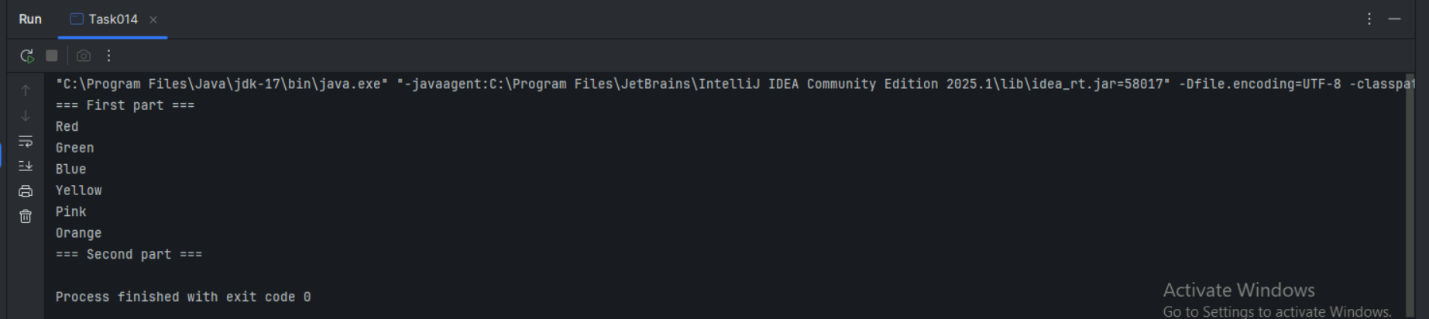
    System.out.println("spliterator 2");

    while( itobj2.tryAdvance( (n) -> { System.out.println(n); } ) );

  }

}





**Task 15**:

What do you understand by a pointer?

**Pointer:** A variable that stores the memory address of another variable, allowing indirect access to data through memory location references.

**How it works:** Instead of storing actual data, pointer holds the address where data is located in memory (e.g., pointer stores address 1000, which contains value 25).

**Usage:** Memory management, dynamic allocation, linked data structures (linked lists, trees), and efficient parameter passing in functions.

**Note:** Java doesn't have explicit pointers like C/C++, but uses object references which work similarly - variables store memory addresses of objects rather than objects themselves.

**Task 16**:

Difference between \* and & in pointers?

**In C/C++ Pointers:**

**& (Address-of operator):**

* Gets the memory address of a variable
* Example: int x = 10; int\* ptr = &x; (ptr stores address of x)

**\* (Dereference operator):**

* Accesses the value stored at the memory address pointed by pointer
* Example: int value = \*ptr; (gets value 10 from address stored in ptr)

**Simple analogy:** & gives you the "house address", \* lets you "enter the house and get contents".

**Note:** Java doesn't use these operators as it handles memory addresses automatically through object references.

**Task 17**:

Wap in c or c++ to implement the use of pointers.

#include <iostream>

using namespace std;

int main() {

// Declare variables

int num = 25;

int another = 50;

// Declare pointer

int\* ptr;

cout << "=== Basic Pointer Operations ===" << endl;

// Assign address of num to pointer

ptr = &num;

cout << "Value of num: " << num << endl;

cout << "Address of num: " << &num << endl;

cout << "Value stored in ptr: " << ptr << endl;

cout << "Value pointed by ptr: " << \*ptr << endl;

cout << "\n=== Modifying through Pointer ===" << endl;

// Modify value through pointer

\*ptr = 100;

cout << "After \*ptr = 100:" << endl;

cout << "Value of num: " << num << endl;

cout << "Value pointed by ptr: " << \*ptr << endl;

cout << "\n=== Pointer Reassignment ===" << endl;

// Point to different variable

ptr = &another;

cout << "Now ptr points to 'another':" << endl;

cout << "Value of another: " << another << endl;

cout << "Value pointed by ptr: " << \*ptr << endl;

cout << "\n=== Array and Pointer ===" << endl;

int arr[] = {10, 20, 30, 40, 50};

int\* arrPtr = arr; // Array name is address of first element

cout << "Array elements using pointer:" << endl;

for(int i = 0; i < 5; i++) {

cout << "arr[" << i << "] = " << \*(arrPtr + i) << endl;

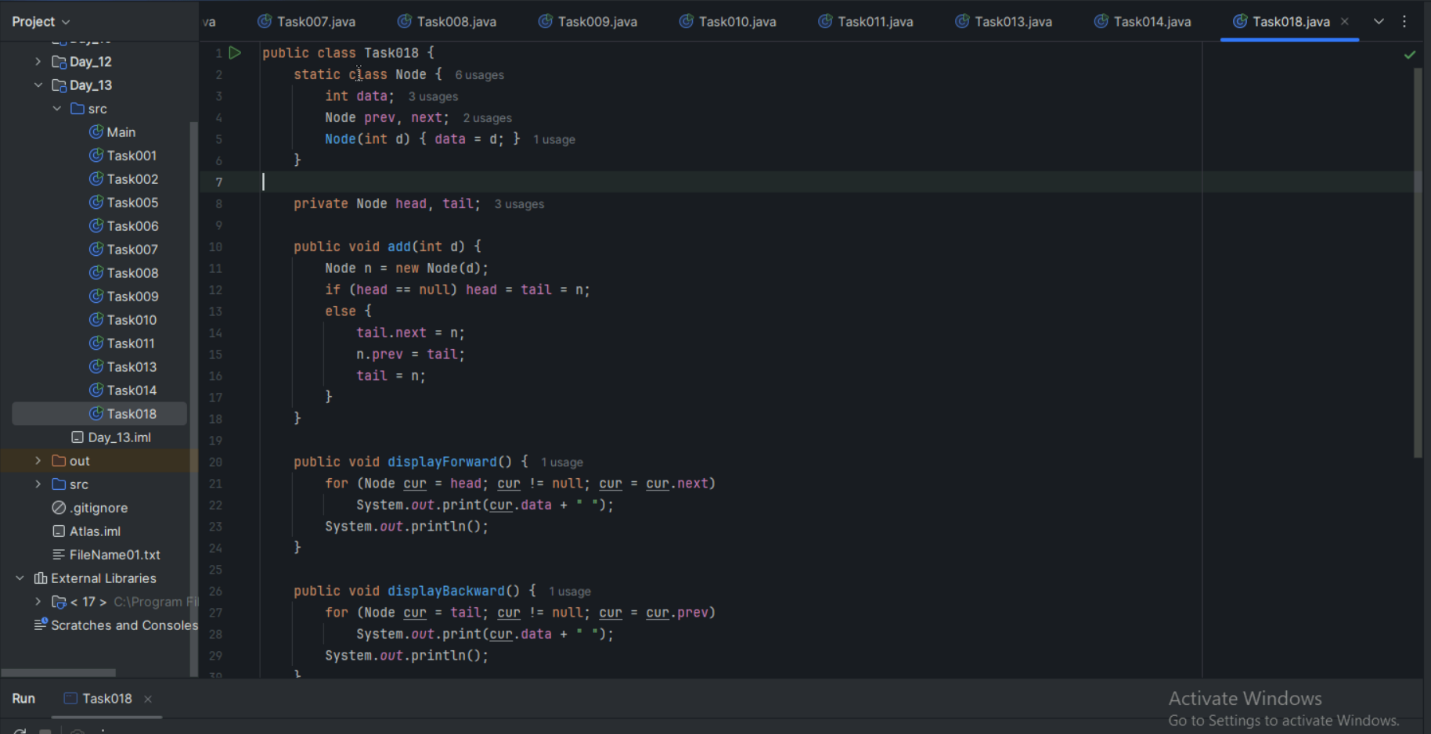
}

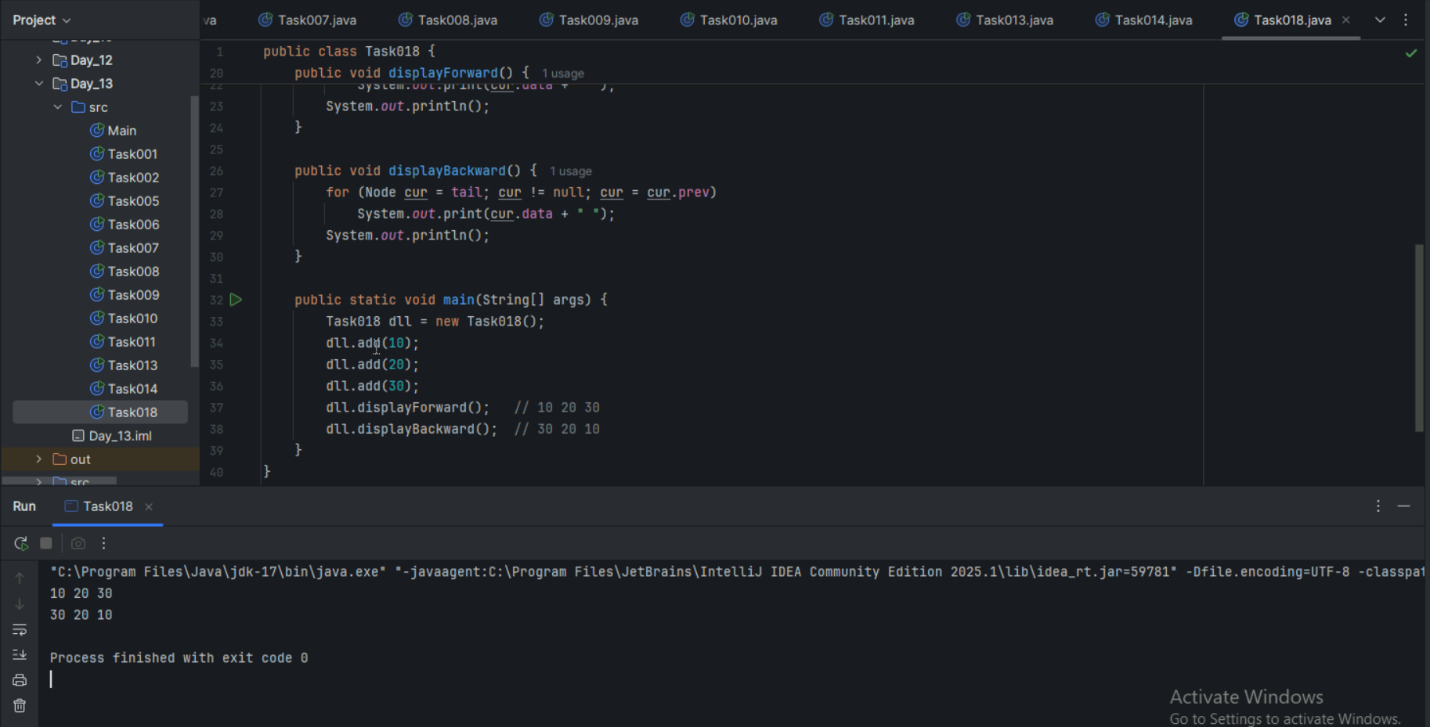
return 0;

}

Task 18:

  Wap to create  a doubly linked list





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

Home Tasks:

1. List Advantages and disadvantages of linked List
2. Applications of Linked list