­­­­­­­Day 13 - 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;

}

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

https://pythontutor.com/render.html#mode=display

By Ahmed.. You can visualise the data structures while executing the code line by line..

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

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) {

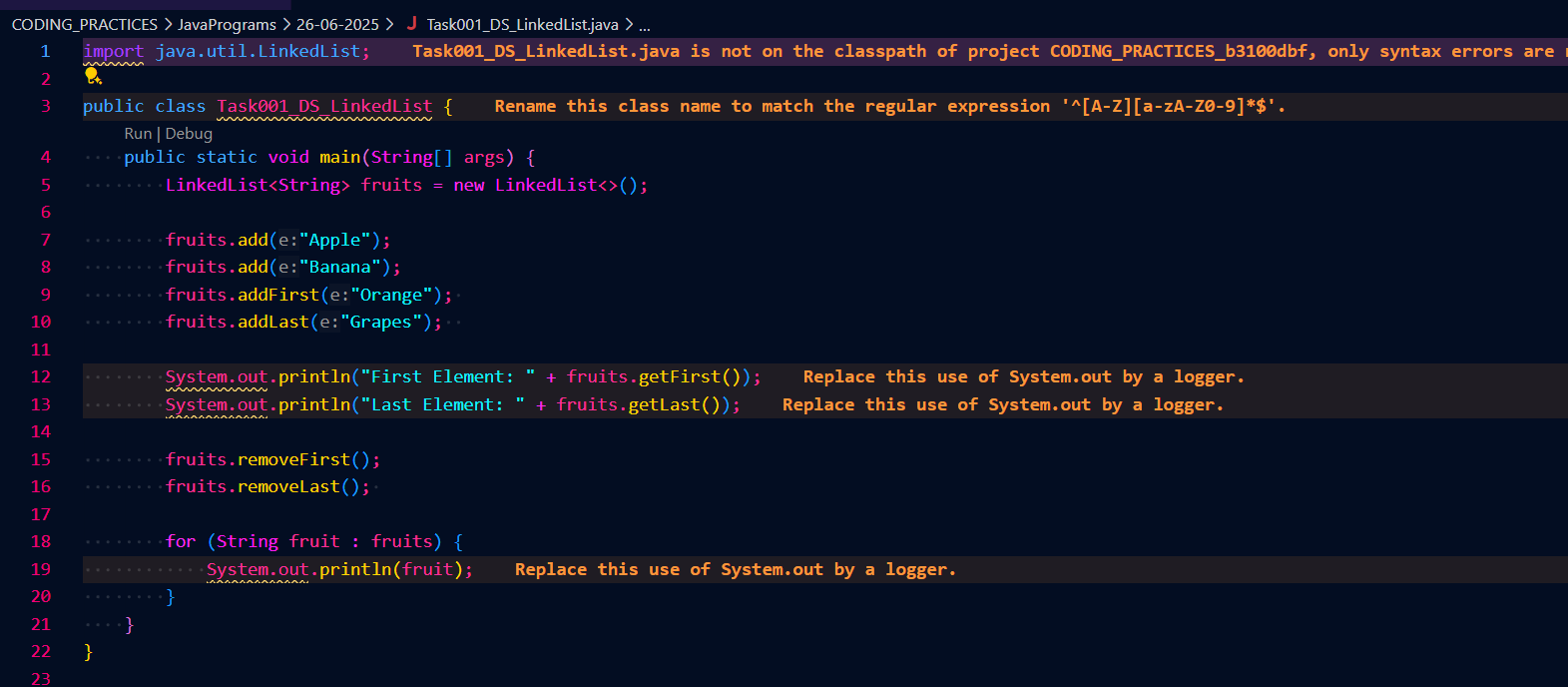
System.out.println(fruit);

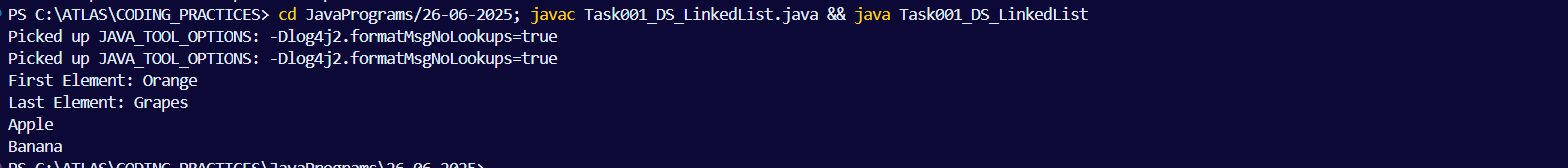
}

}

}

Solution :





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

Task 002 :

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

B)Try to add element at the end of the list

C) Add the element

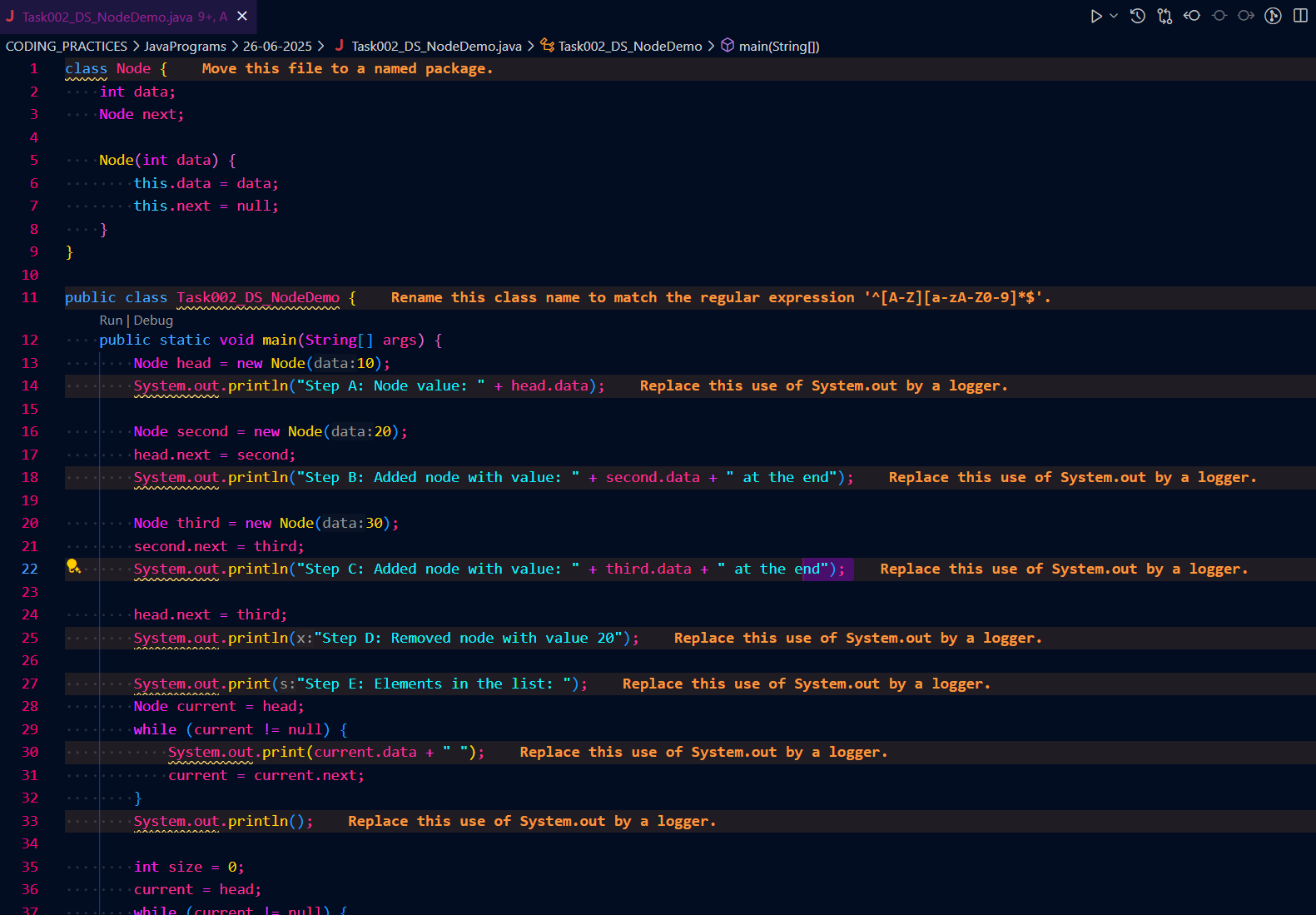
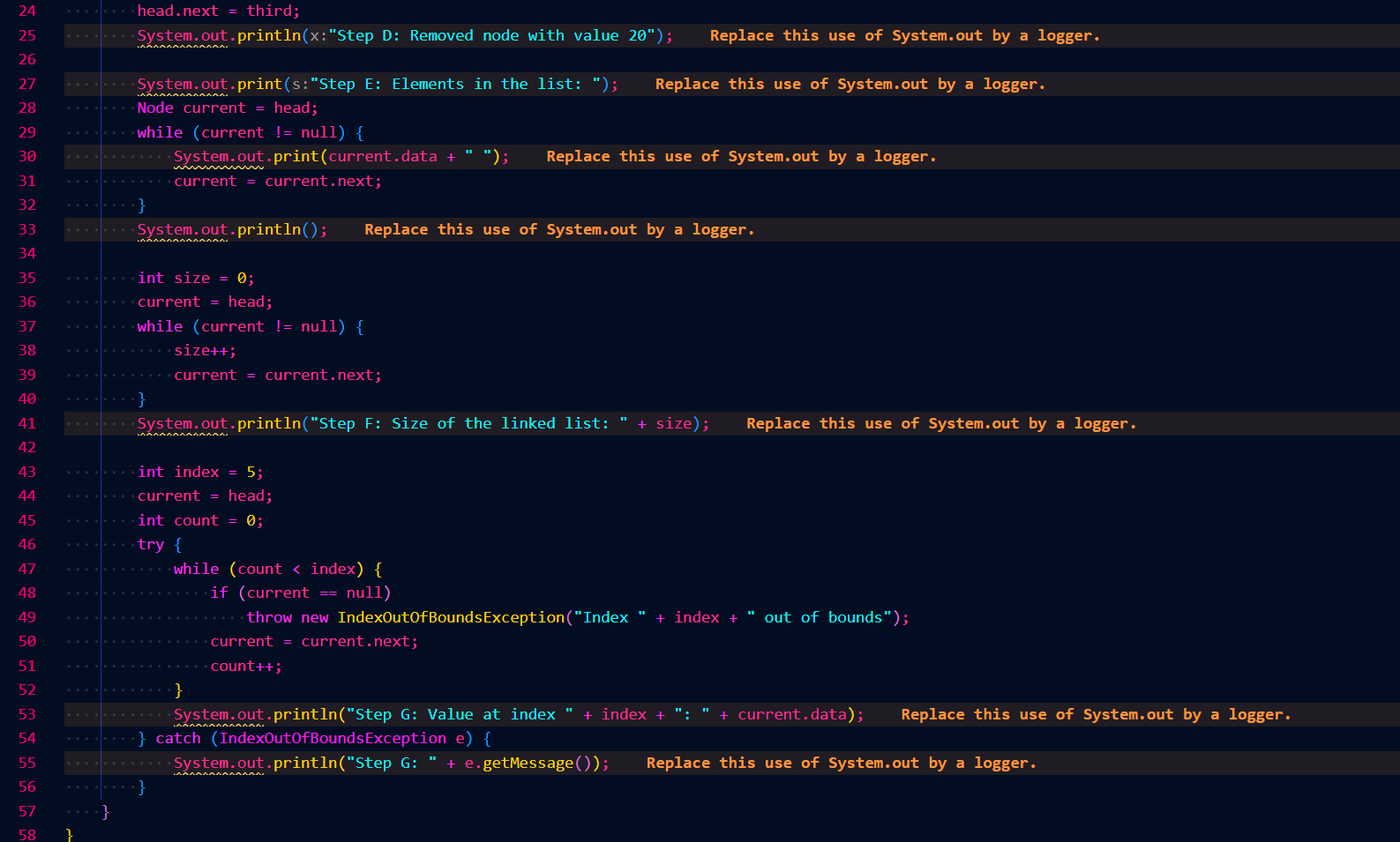
D) Remove the node

E) Display all the elements of the node

F) Find size of the linked list

G) Index out of bounds

Solution :

Home Task :

class Node<T> {

T data;

Node<T> next;

public Node(T data) {

this.data = data;

this.next = null;

}

}

public class CustomLinkedList<T> {

private Node<T> head;

private int size = 0;

public void add(T data) {

Node<T> newNode = new Node<>(data);

if (head == null) {

head = newNode;

} else {

Node<T> current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

size++;

}

public void addFirst(T data) {

Node<T> newNode = new Node<>(data);

newNode.next = head;

head = newNode;

size++;

}

public T removeFirst() {

if (head == null) {

throw new NoSuchElementException("List is empty");

}

T removedData = head.data;

head = head.next;

size--;

return removedData;

}

public T get(int index) {

checkBounds(index);

Node<T> current = head;

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

current = current.next;

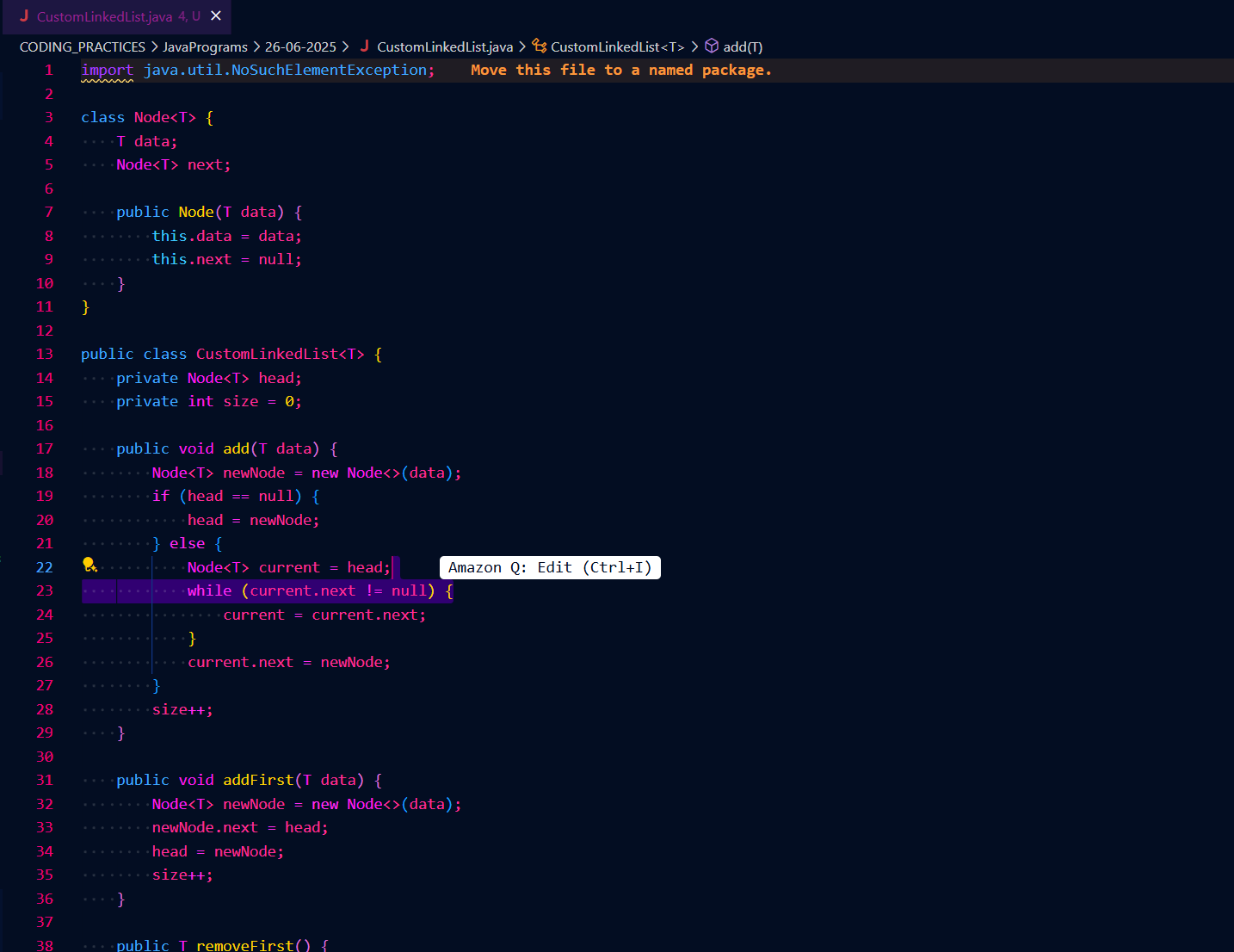
}

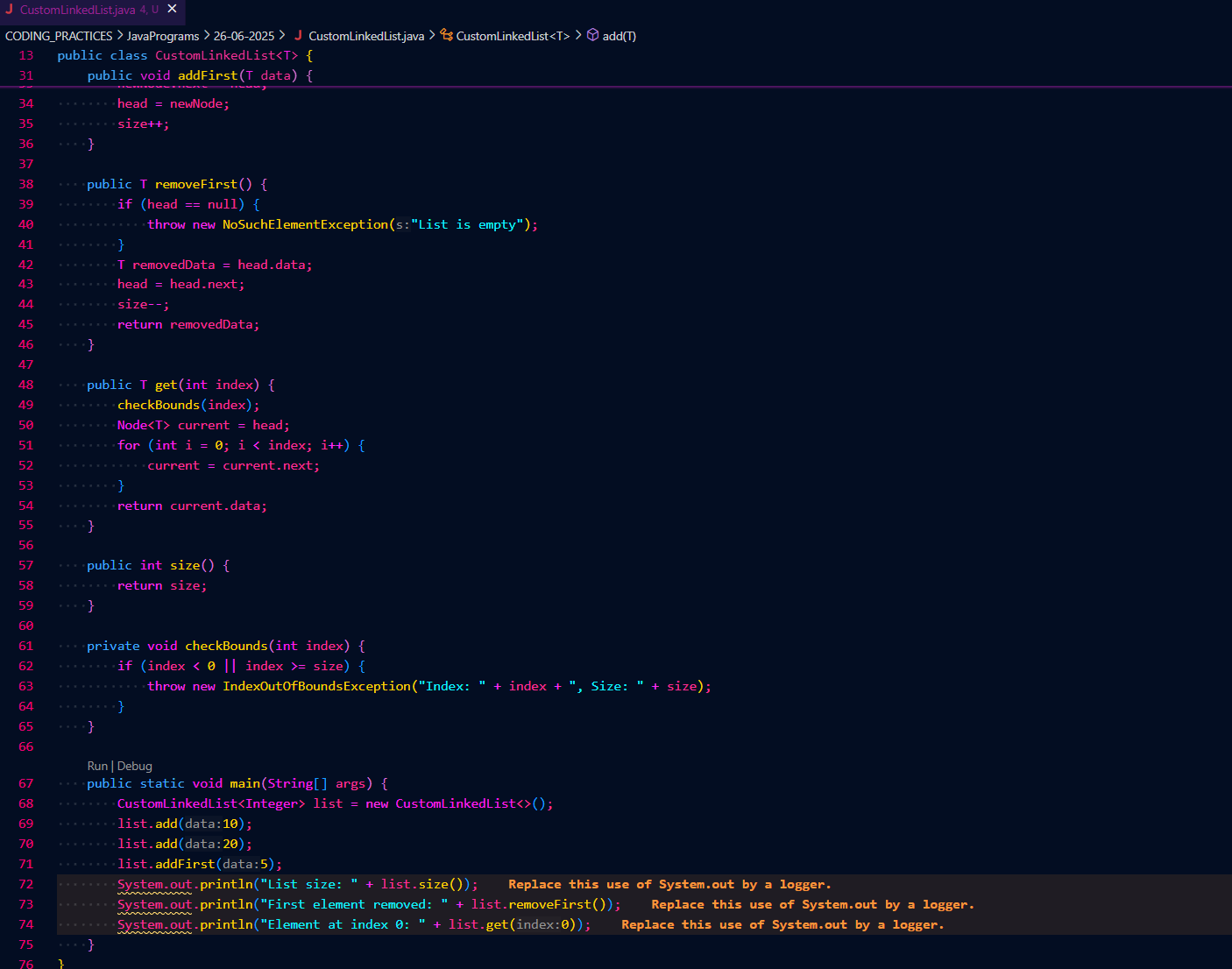
return current.data;

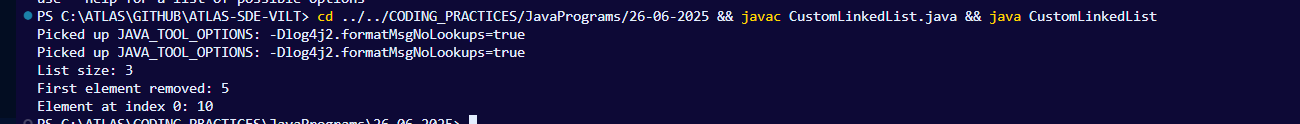
}

public int size() {

return size;

Solution : 





Task 3 : List down the methods of linked lists..

Solution : Summary of common linked list methods (for singly, doubly, and circular linked lists) with one-line descriptions:

Singly Linked List

add(T data): Add element at the end.

addFirst(T data): Add element at the beginning.

removeFirst(): Remove and return the first element.

remove(int index): Remove element at a specific index.

get(int index): Get element at a specific index.

set(int index, T data): Update element at a specific index.

contains(T data): Check if an element exists.

size(): Return the number of elements.

isEmpty(): Check if the list is empty.

clear(): Remove all elements.

indexOf(T data): Get the index of an element.

Doubly Linked List (in addition to above)

addLast(T data): Add element at the end (alias for add).

addBefore(Node node, T data): Add element before a given node.

addAfter(Node node, T data): Add element after a given node.

removeLast(): Remove and return the last element.

getPrevious(Node node): Get the previous node.

getNext(Node node): Get the next node.

Circular Linked List (in addition to singly)

add(T data): Add element, maintaining circular link.

remove(T data): Remove element, handling circular reference.

getNext(Node node): Get next node, loops to head if at end.

isCircular(): Check if the list is circular.

Task 4 : What are the operations of data structures.. one liner?(Ex : Traversing,Insertion,Deletion ,Searching,Sorting)

Solution :

Traversing: Accessing each element of the data structure exactly once.

Insertion: Adding a new element to the data structure.

Deletion: Removing an existing element from the data structure.

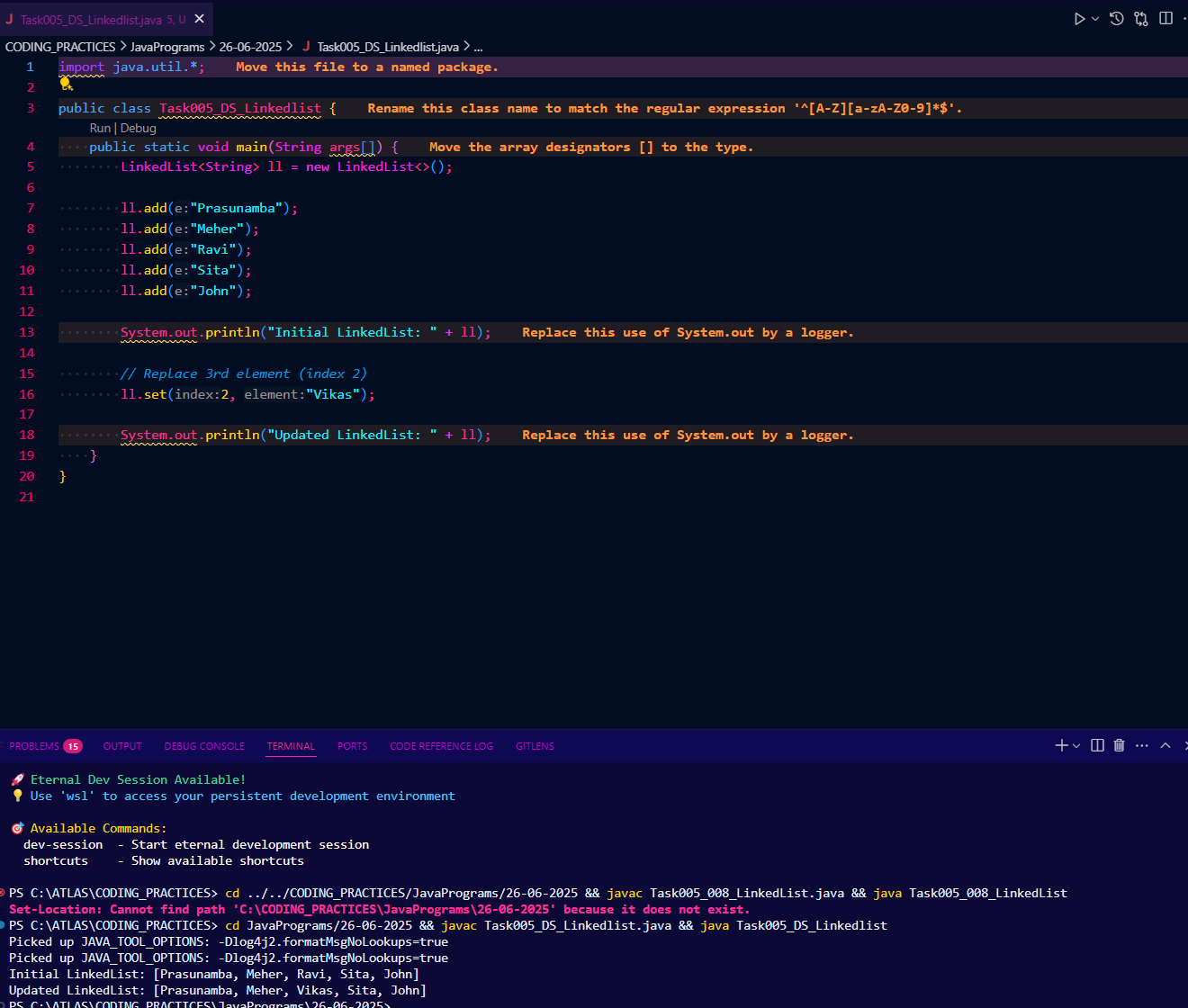
Searching: Finding the location of a specific element.

Sorting: Arranging elements in a particular order (ascending or descending).

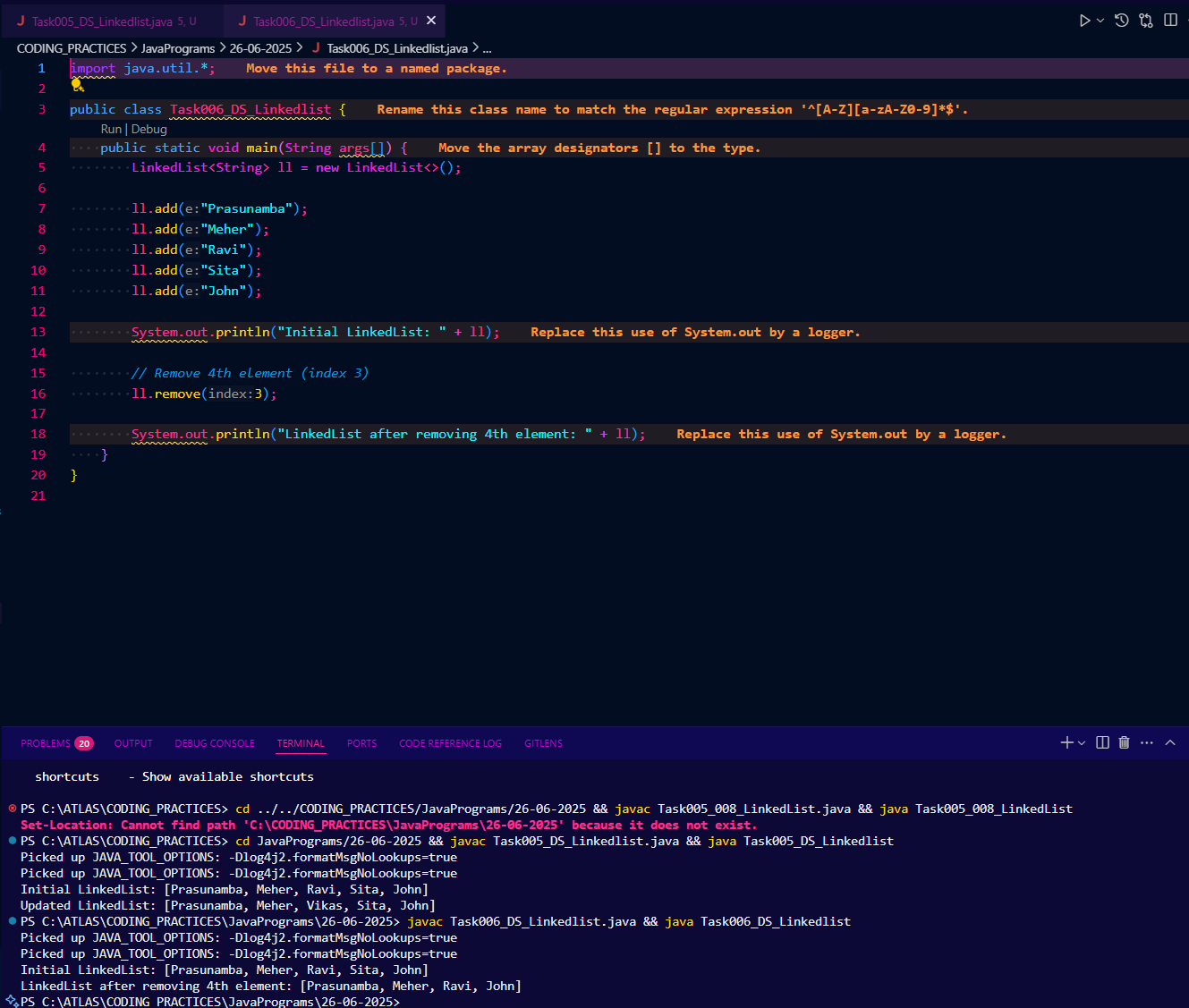
Merging – Combining two data structures into one.

Updating – Modifying the value of an existing element.

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

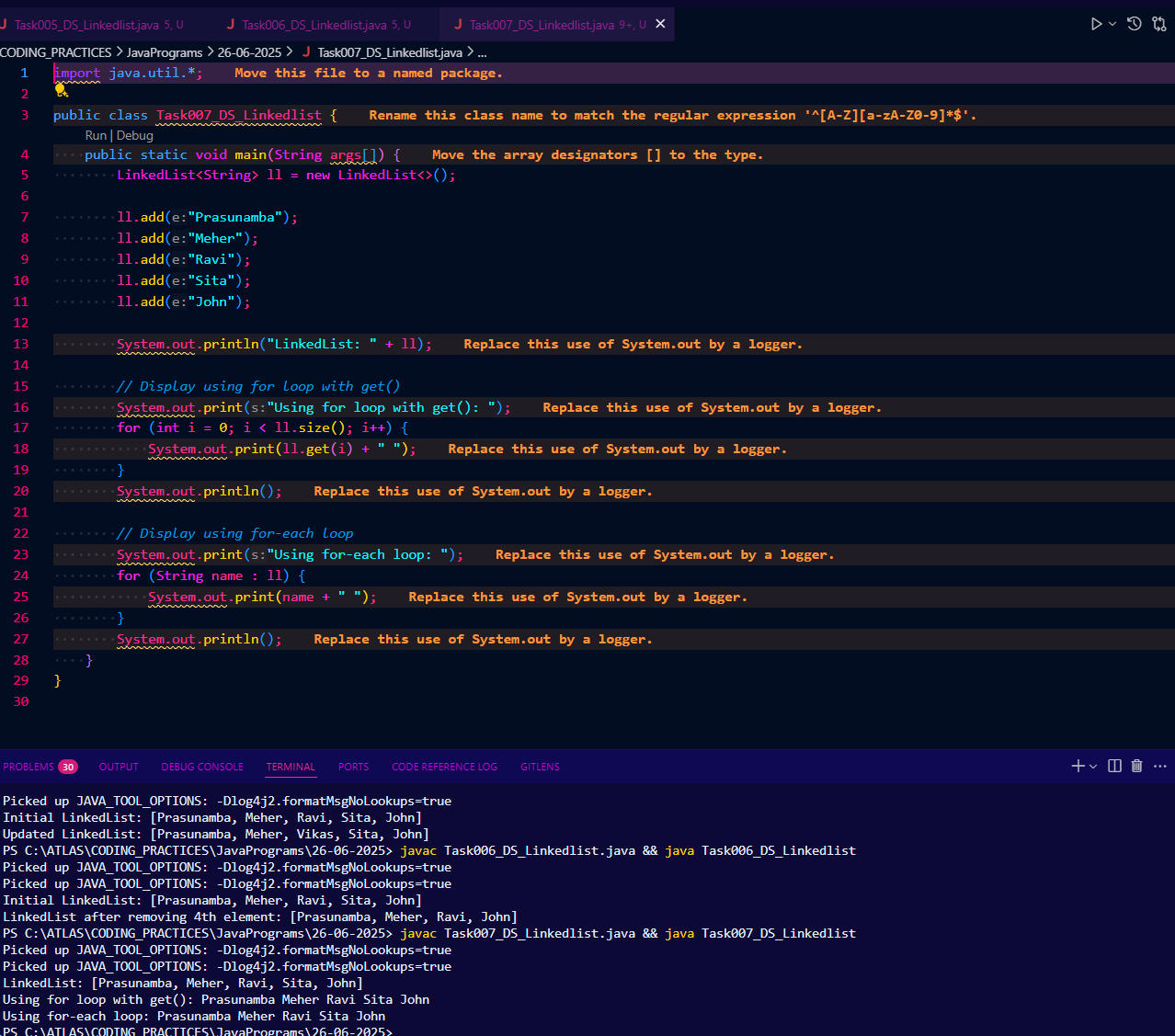
Solution : 

Task 6: Wap to create a linked list to add 5 elements and remove any element and display

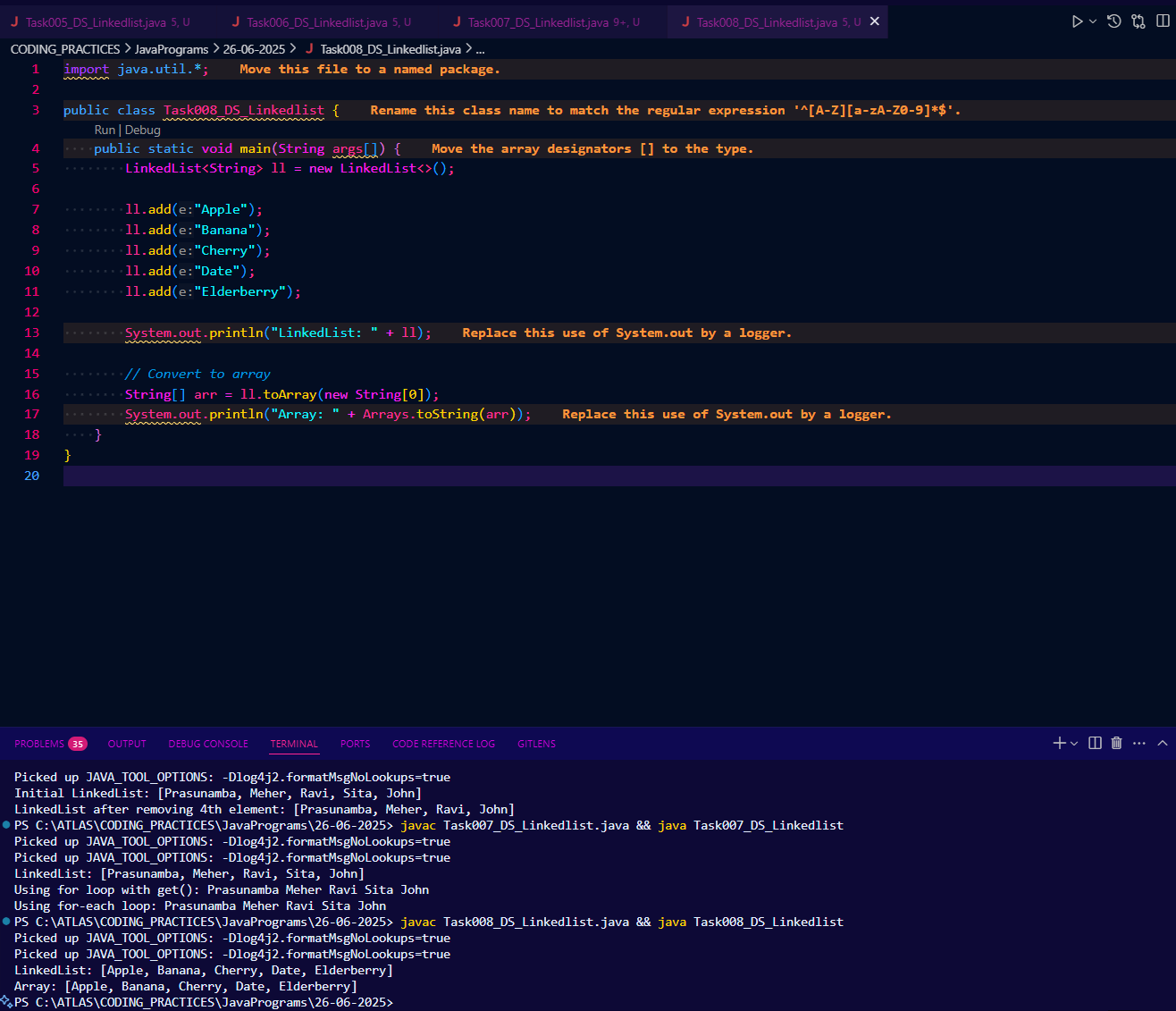
Solution : 

Task 7: Wap to create a linked list to add 5 elements and display the list using for (use get() ) and for each loops

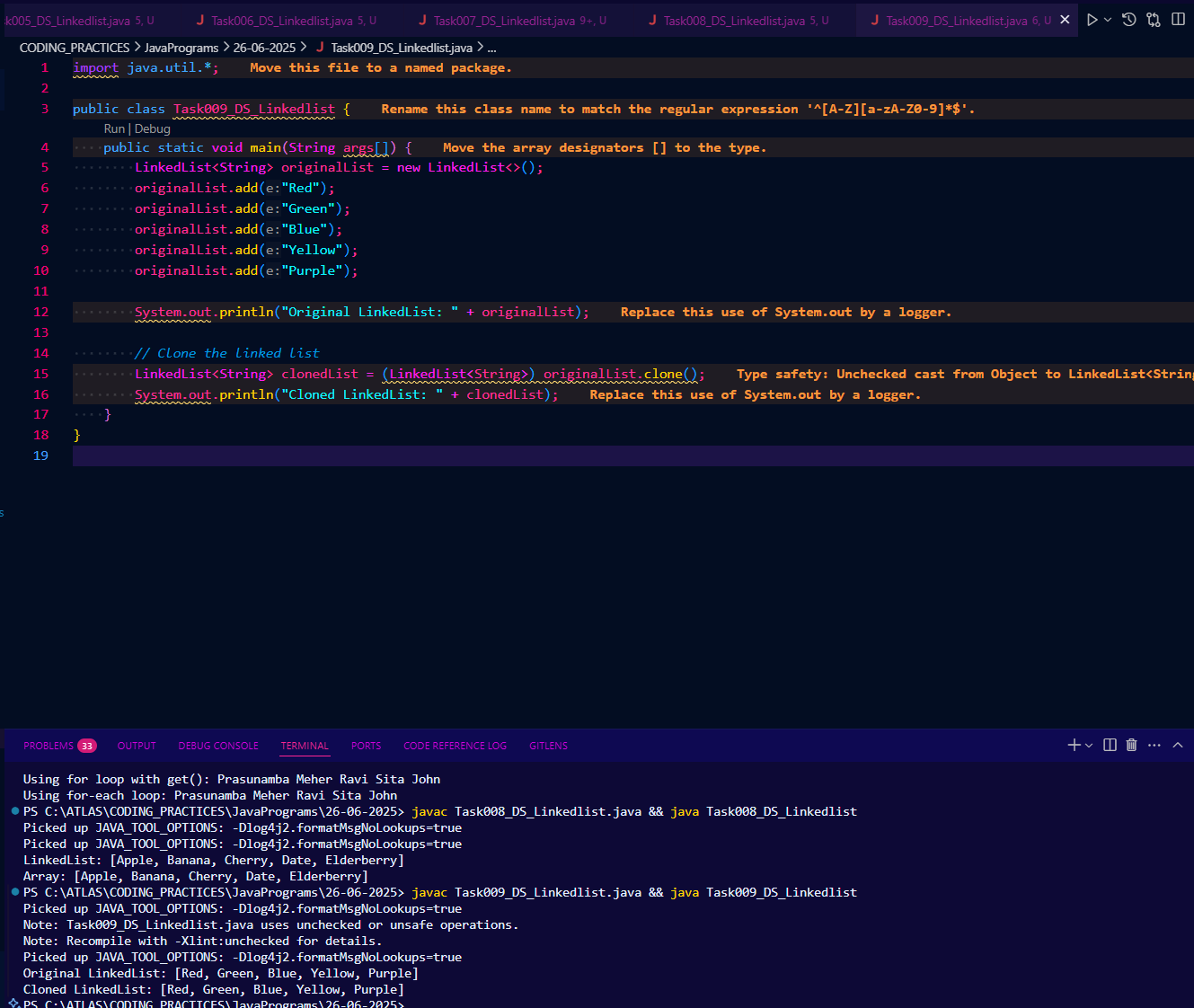
Solution :



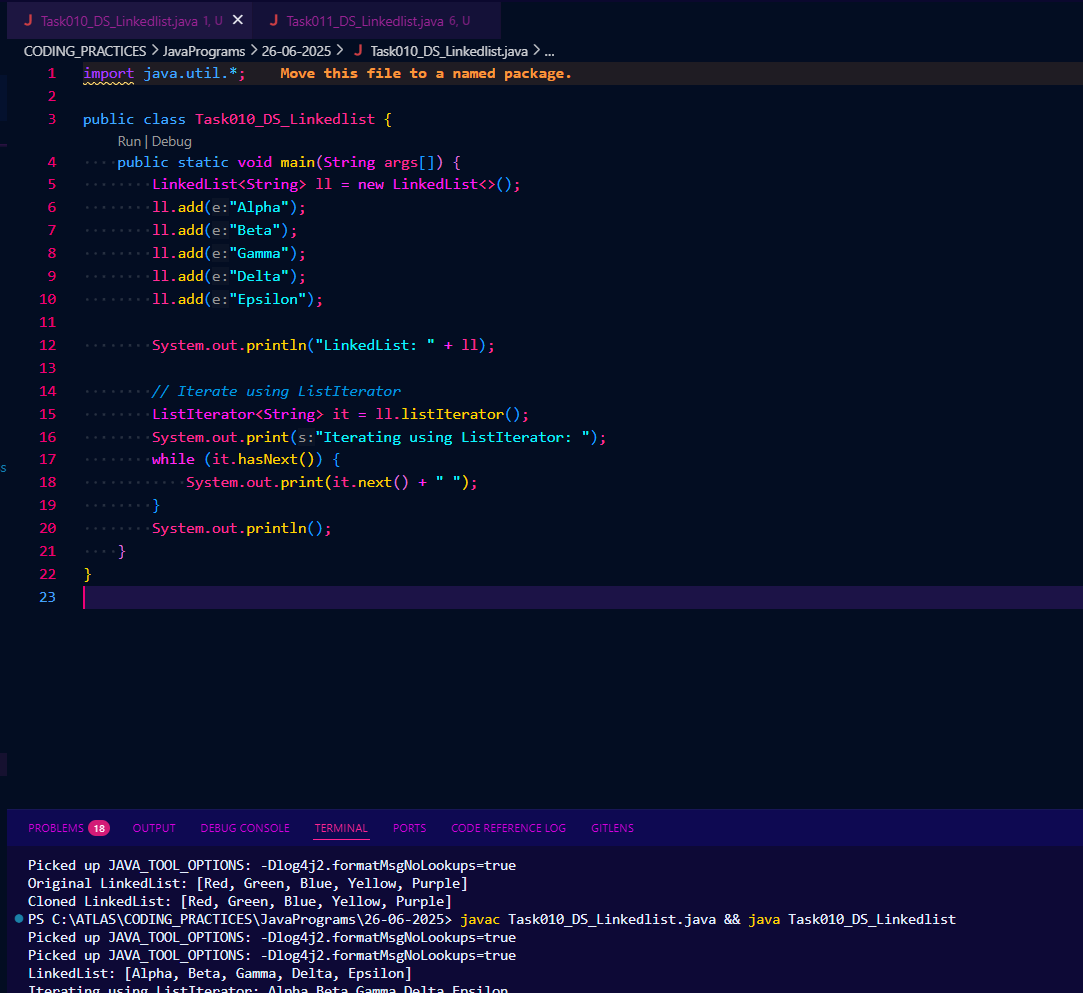
Task 8: Create a linked list and few items and convert it into an array

Solution : 

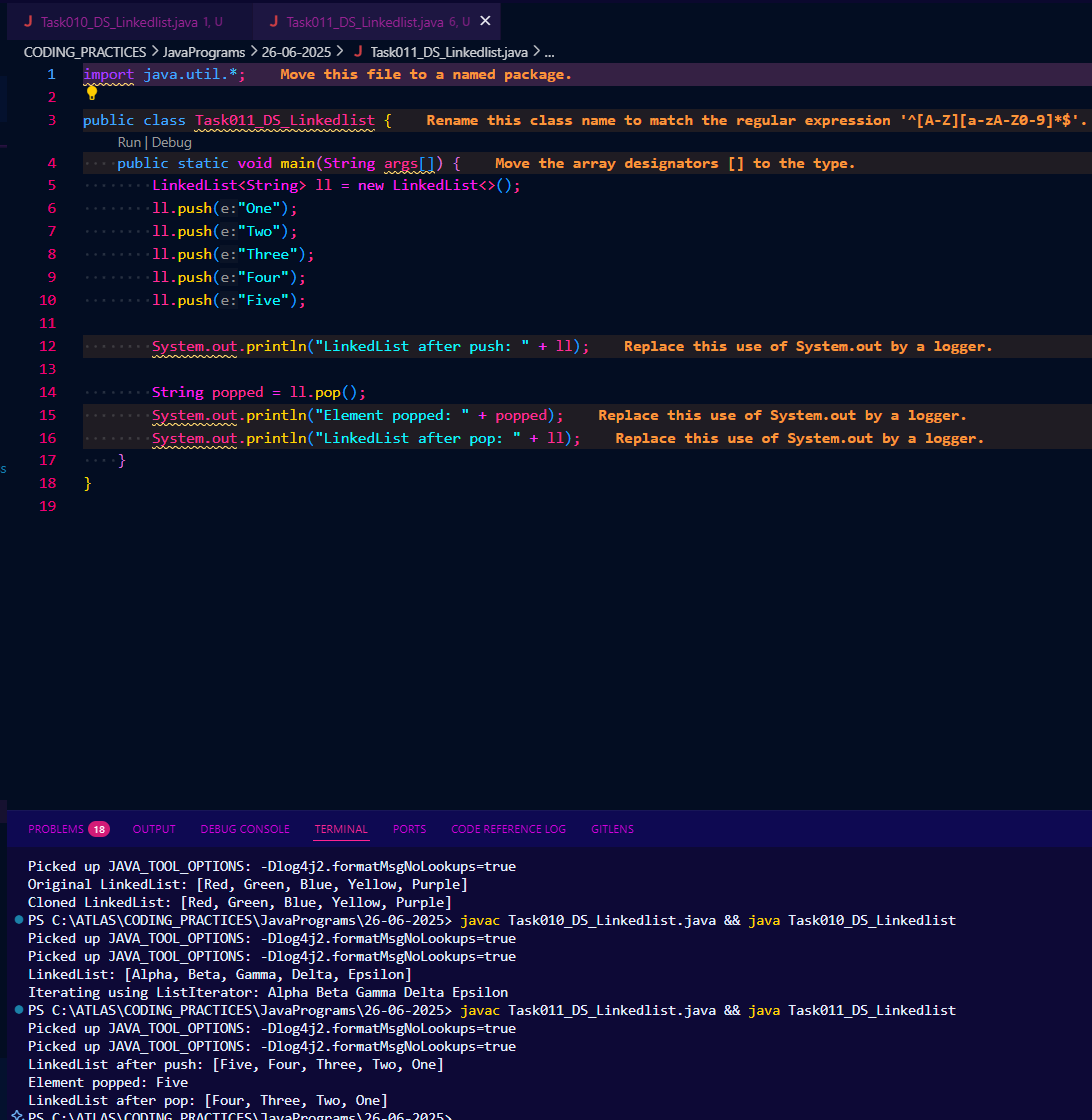
Task 9: Create a linked list add few items and clone the 1st linked list with the 2nd linked list

Solution : 

Task 10: Create a linked list and iterate the values using ListIterator class in util package

Solution : 

Task 11: Create a linked list and use push and pop methods.

Solution : 

Task 12: Difference between Iterator and splitIterator

Solution : Difference between Iterator and Spliterator

Iterator:

- Used for sequential traversal of a collection.

- Cannot split or partition the collection.

- Methods: hasNext(), next(), remove().

- Suitable for single-threaded, simple iteration.

Spliterator:

- Can traverse and also split the collection for parallel processing.

- Methods: tryAdvance(), trySplit(), forEachRemaining().

- Used with Java Streams for efficient parallelism.

- Suitable for both sequential and parallel iteration.

Summary:

Use Iterator for simple, sequential access.

Use Spliterator when you need to split the collection for parallel processing or advanced traversal.

Task 13:

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();

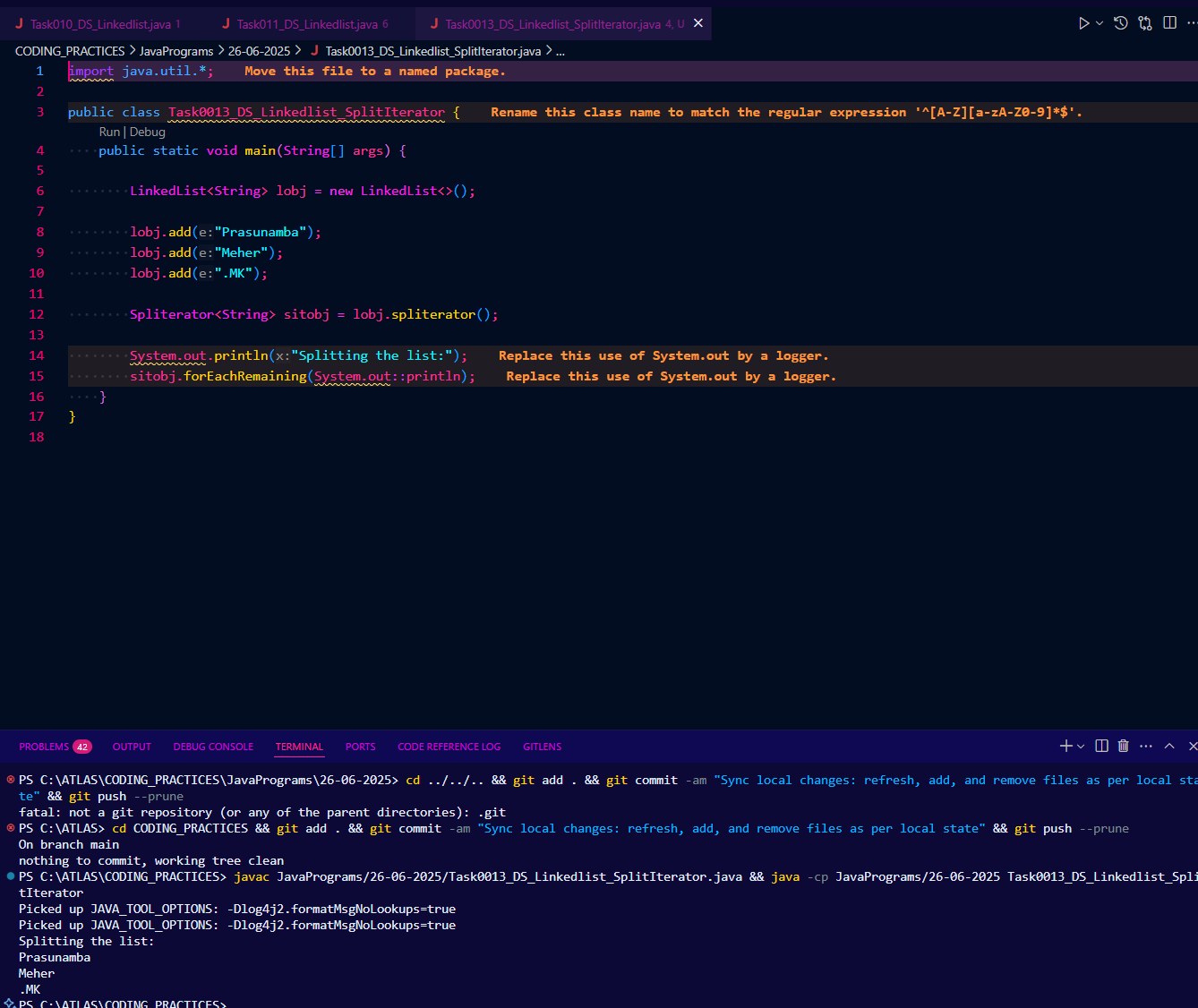
System.out.println("Splitting the list:");

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

}

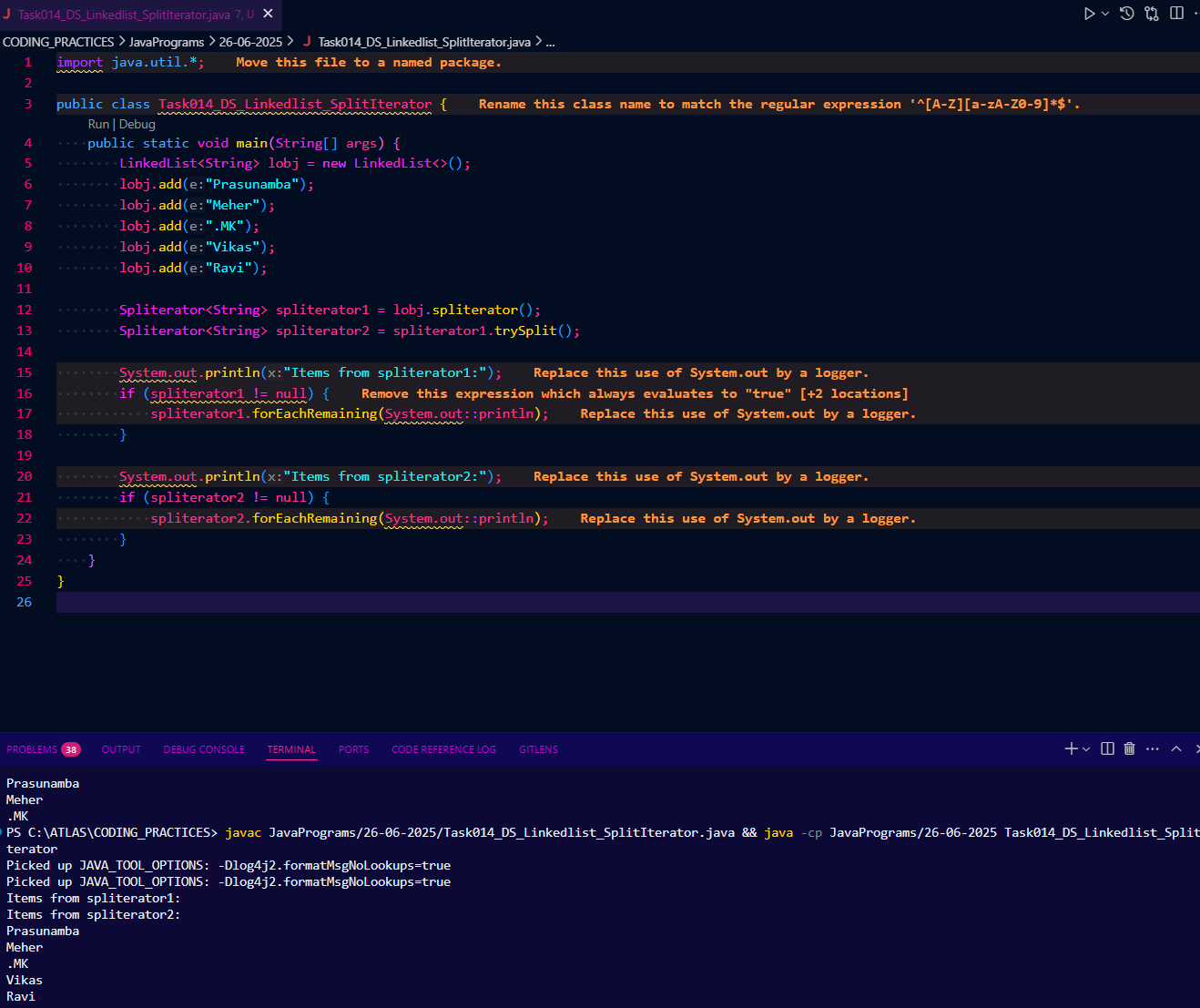
}

Solution :

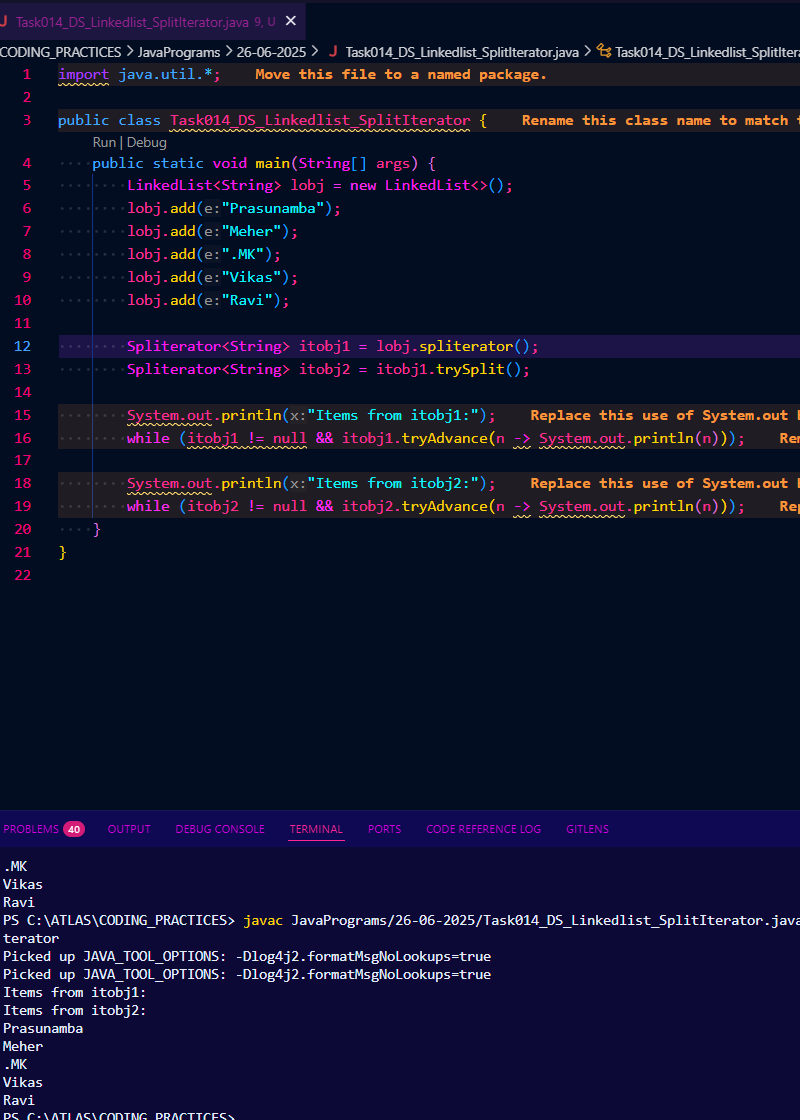


Task 14: Create a linked list and display items into 2 lists using split iterator

Solution :



With tryadvance :



Task 15: What do you understand by a pointer?

Solution :

A pointer is a variable that stores the memory address of another variable, rather than storing a value directly. Think of it like a house address - instead of containing the house itself, it contains the location where you can find the house.

Key characteristics of pointers:

Memory Address Storage: Pointers hold the address in memory where data is stored, not the actual data itself.

Indirect Access: To access the value that a pointer points to, you need to "dereference" it using the \* operator.

Data Type Specific: Pointers are typed - an int\* pointer can only point to integer variables, a char\* pointer points to character variables, etc.

Dynamic Memory Management: Pointers enable dynamic allocation and deallocation of memory during program execution.

Efficient Parameter Passing: Instead of copying large data structures, you can pass their addresses, saving memory and time.

Task 16: Difference between \* and & in pointers?

Solution :

The \* and & operators serve opposite purposes when working with pointers:

The & Operator (Address-of)

Purpose: Gets the memory address of a variable

Usage: &variable\_name

Returns: The memory address where the variable is stored

Called: Address-of operator or reference operator

The \* Operator (Dereference)

Purpose: Accesses the value stored at a memory address

Usage: \*pointer\_name

Returns: The actual value stored at that memory location

Called: Dereference operator or indirection operator

Key Differences:

& creates a pointer (gets address)

\* uses a pointer (gets value from address)

& is used in pointer assignment

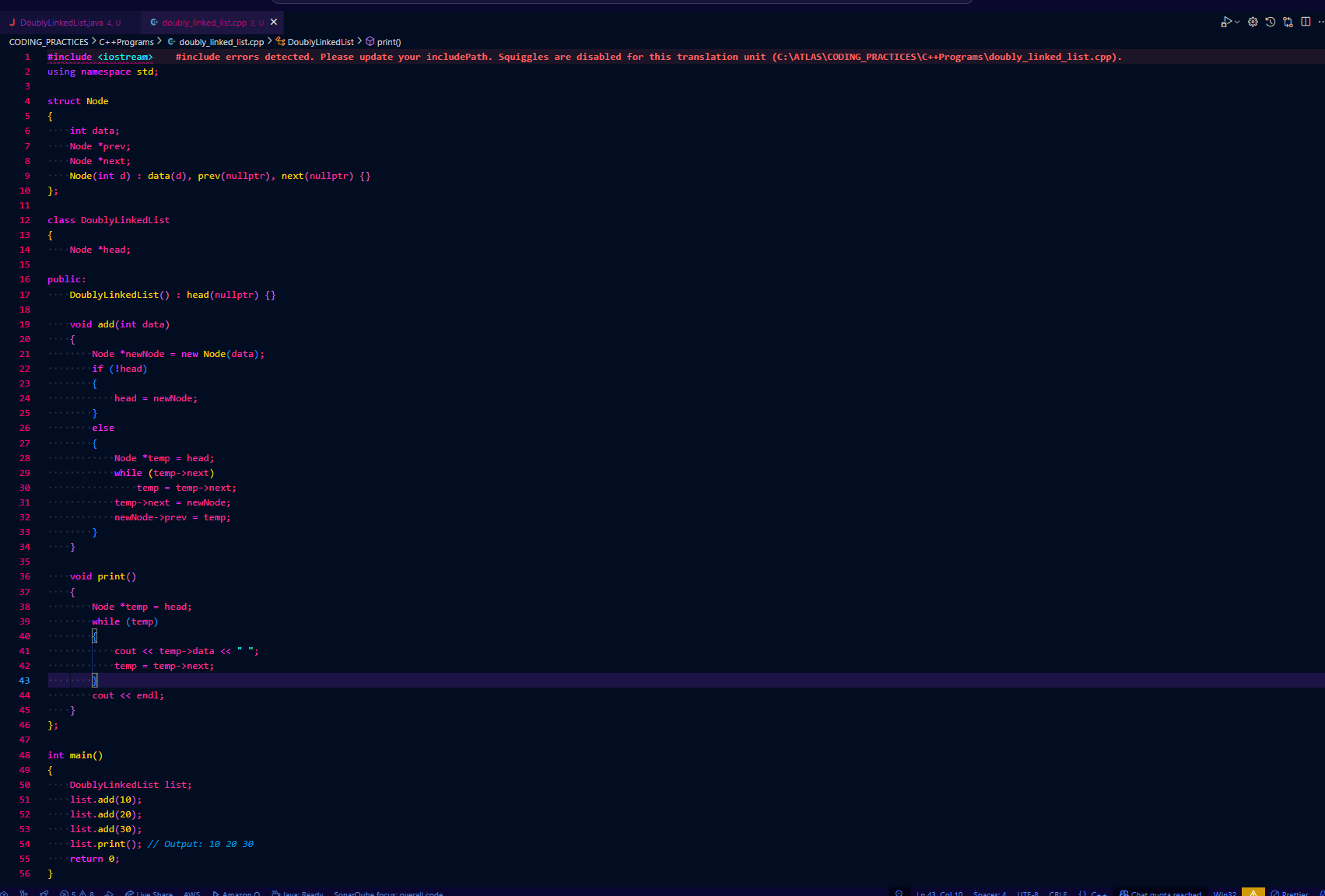
\* is used in pointer declaration and dereferencing

They are essentially inverse operations

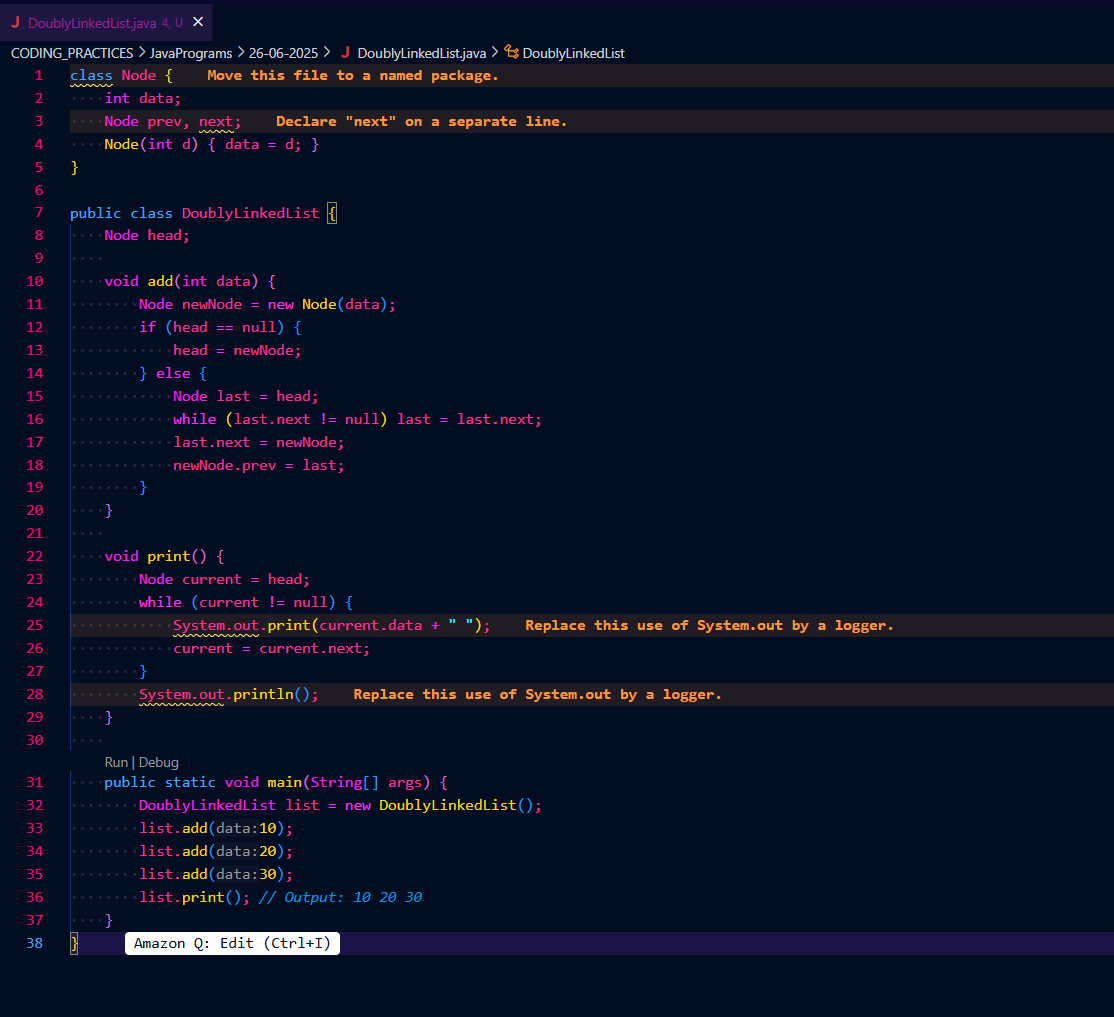
This relationship allows pointers to provide both indirect access to variables and efficient memory management in programming languages like C and C++.

Task 17: Wap in c or c++ to implement the use of pointers.

Solution :



Task 18: Wap to create a doubly linked list

Solution : 

Home Tasks:

Q) List Advantages and disadvantages of linked List

A) Here are the main advantages and disadvantages of linked lists:

\*\*Advantages:\*\*

- Dynamic size: Can grow or shrink at runtime, unlike arrays.

- Easy insertion/deletion: Adding or removing elements (especially at the beginning) is efficient (O(1) if pointer is known).

- No memory wastage: No need to pre-allocate memory; nodes are allocated as needed.

- Efficient memory utilization: Useful when the number of elements is unknown or changes frequently.

\*\*Disadvantages:\*\*

- More memory usage: Each node stores extra pointers (next, and possibly previous).

- No random access: Accessing an element by index requires traversal from the head (O(n) time).

- More complex code: Implementation and pointer management are more error-prone than arrays.

- Poor cache locality: Nodes may be scattered in memory, leading to slower access compared to arrays.

Q) Applications of Linked list

A) Here are some common applications of linked lists:

Implementation of stacks and queues.

Dynamic memory allocation (e.g., free lists in memory management).

Undo functionality in editors (history as a linked list).

Adjacency lists in graph data structures.

Polynomial arithmetic and sparse matrix representation.

Navigation systems (forward/backward in browsers, music playlists).

Hash table chaining for collision resolution.