Day 13 - 26th June 2025

Name: ARAVIND KASANAGOTTU

ID:MVSNARAV

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();

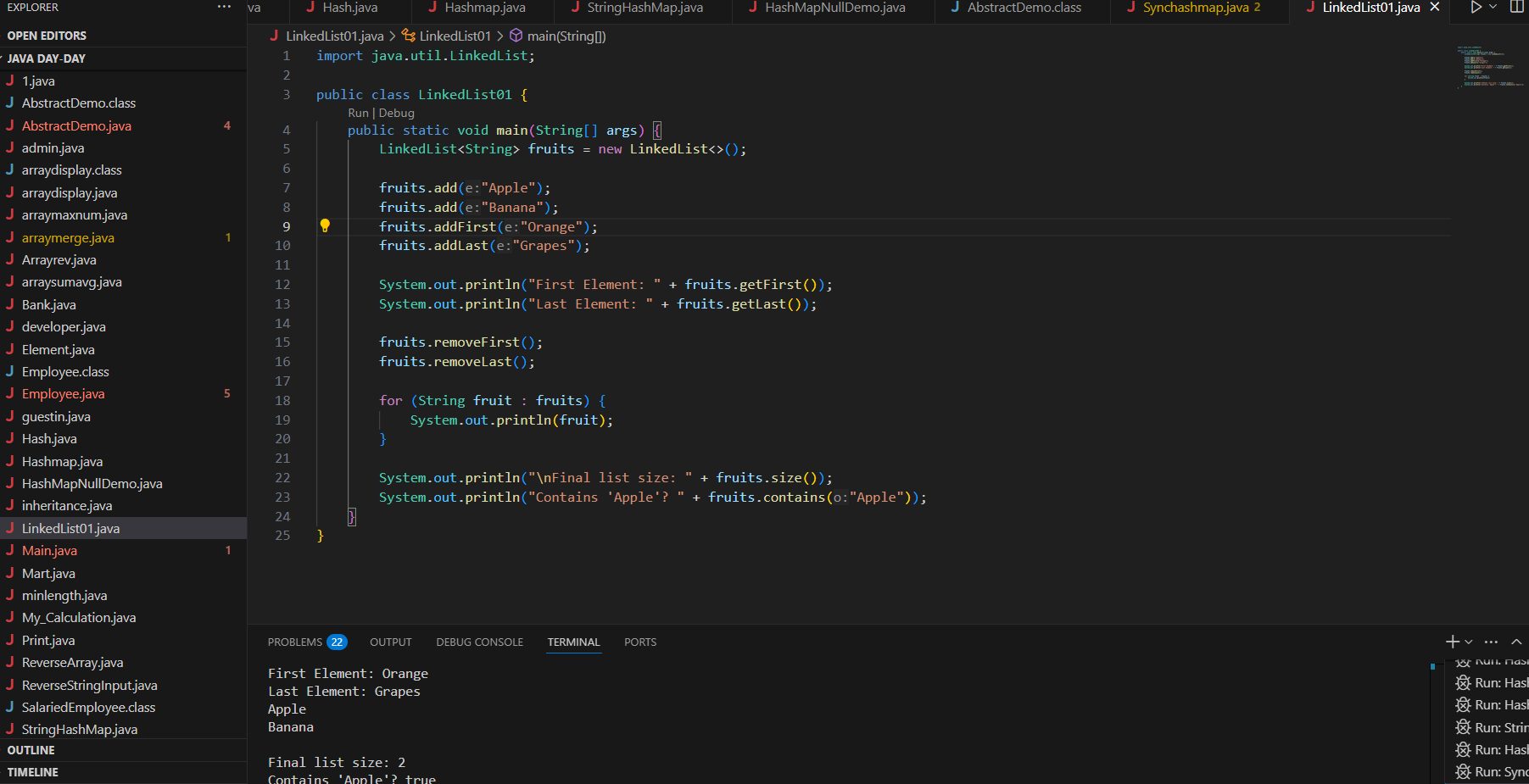
        for (String fruit : fruits) {

            System.out.println(fruit);

        }

    }

}



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

Task 002

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

~~Create a node~~

Try to add element at the end of the list

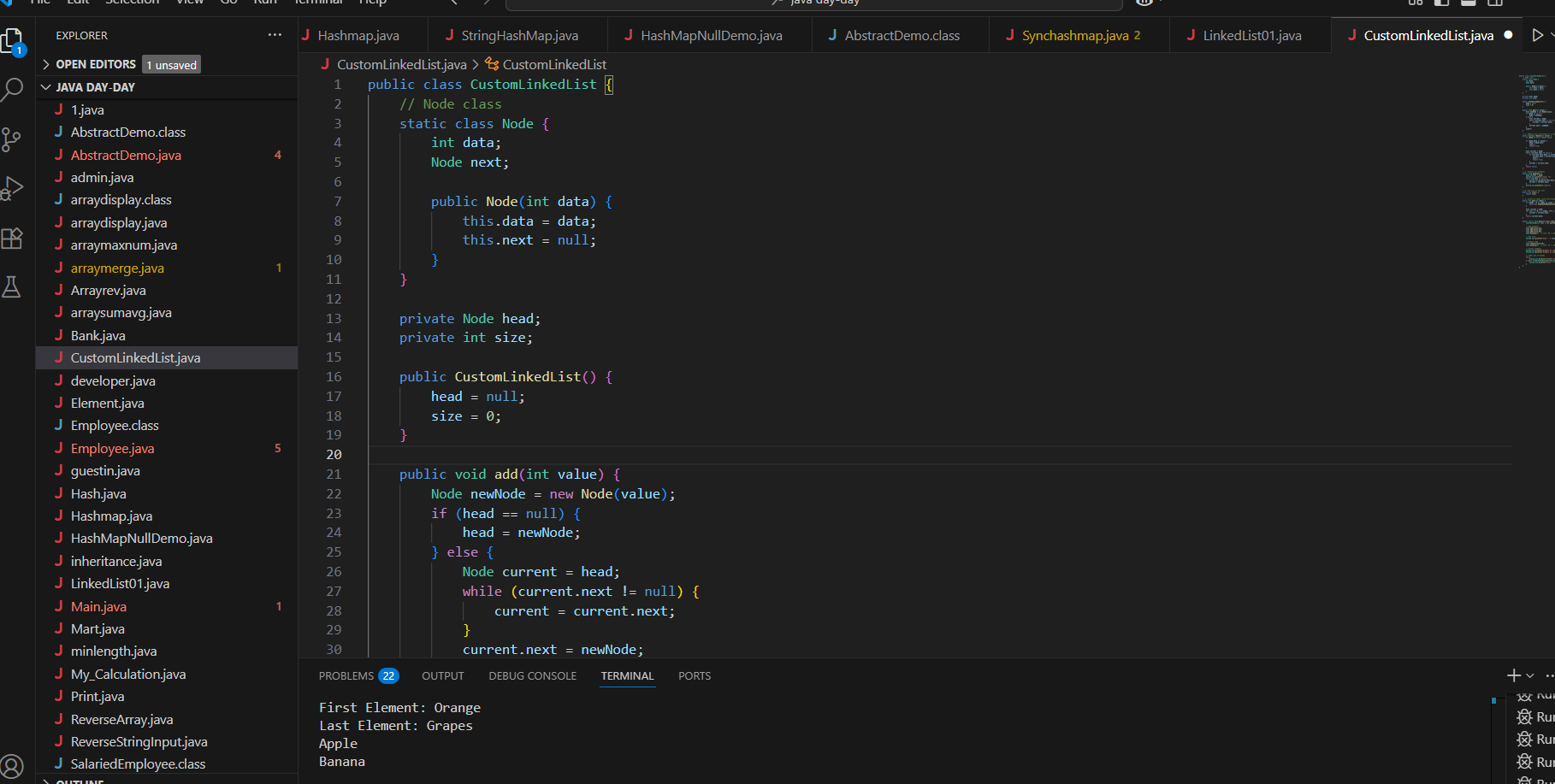
Add the element

Remove the node

Display all the elements of the node

Find size of the linked list

Index out of bounds



11.35 to 11.40

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;

    }

    private void checkBounds(int index) {

        if (index < 0 || index >= size) {

            throw new IndexOutOfBoundsException("Index out of bounds");

        }

    }

}

public class Task002\_DS\_CustomLinkedList {

    public static void main(String[] args) {

        CustomLinkedList<String> liobj = new CustomLinkedList<>();

        liobj.add("Anitha");

        liobj.add("Verma");

        liobj.addFirst("Jack");

        System.out.println("First Element: " + liobj.get(0));

        System.out.println("Size: " + liobj.size());

        liobj .removeFirst();

        System.out.println("First Element after removal: " + liobj.get(0));

        System.out.println("Size after removal: " + liobj.size());

    }

}

Task 3

List down the methods of linked lists..

1. **Element Insertion**:
   * add(E e): Appends to end
   * addFirst(E e): Inserts at beginning
   * addLast(E e): Appends to end
   * add(int index, E element): Inserts at position
2. **Element Removal**:
   * remove(): Removes first element
   * removeFirst(): Removes first element
   * removeLast(): Removes last element
   * remove(int index): Removes by position
   * remove(Object o): Removes first occurrence
3. **Element Access**:
   * get(int index): Returns element by position
   * getFirst(): Returns first element
   * getLast(): Returns last element
   * peek(): Retrieves head (no removal)
   * peekFirst()/peekLast(): End element access
4. **Queue Operations**:
   * offer(E e): Adds to end (queue-style)
   * poll(): Removes and returns head
   * element(): Retrieves head
5. **Stack Operations**:
   * push(E e): Adds to head (stack push)
   * pop(): Removes from head (stack pop)
6. **Search & Status**:
   * size(): Returns element count
   * contains(Object o): Checks existence
   * indexOf(Object o): Finds first position
   * isEmpty(): Checks if empty
7. **Bulk Operations**:
   * clear(): Removes all elements
   * toArray(): Converts to array

Task 4:

What are the operations of data structures.. I liner

Traversing

Insertion

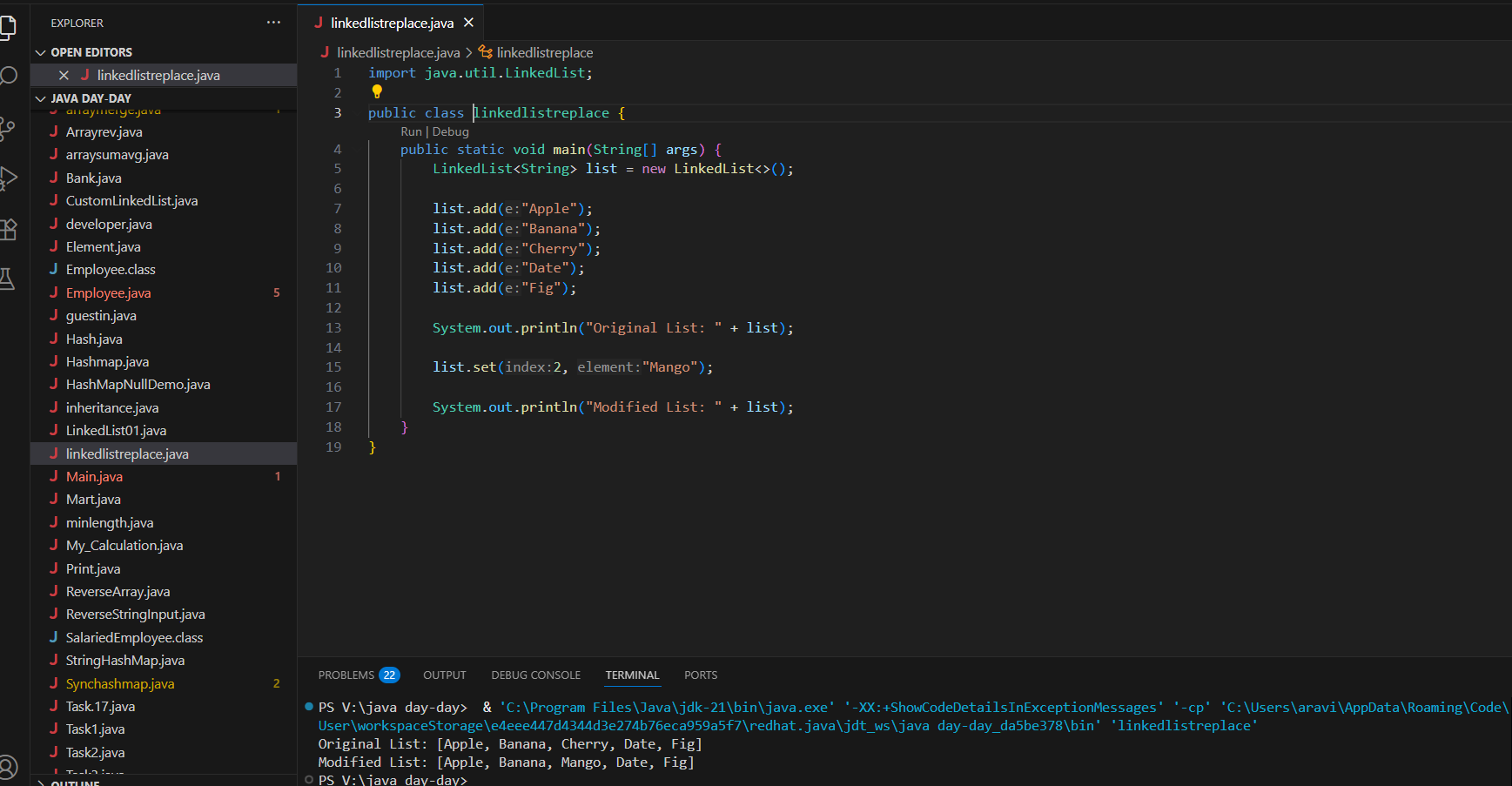
Deletion

Searching

Sorting

Task 5:

Wap to create a linked list add 5 elements to it and replace 3 rd 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);

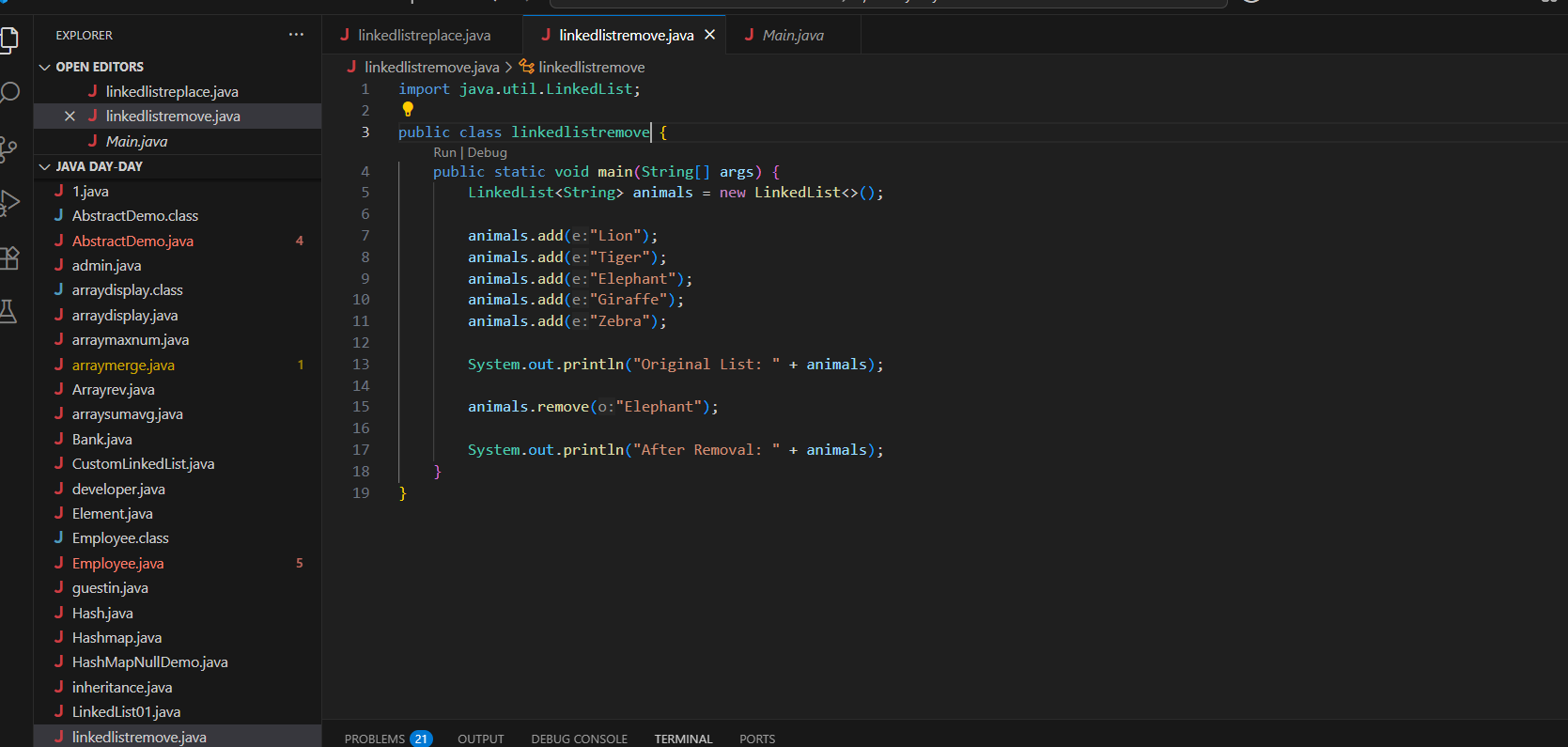
    }

}

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");

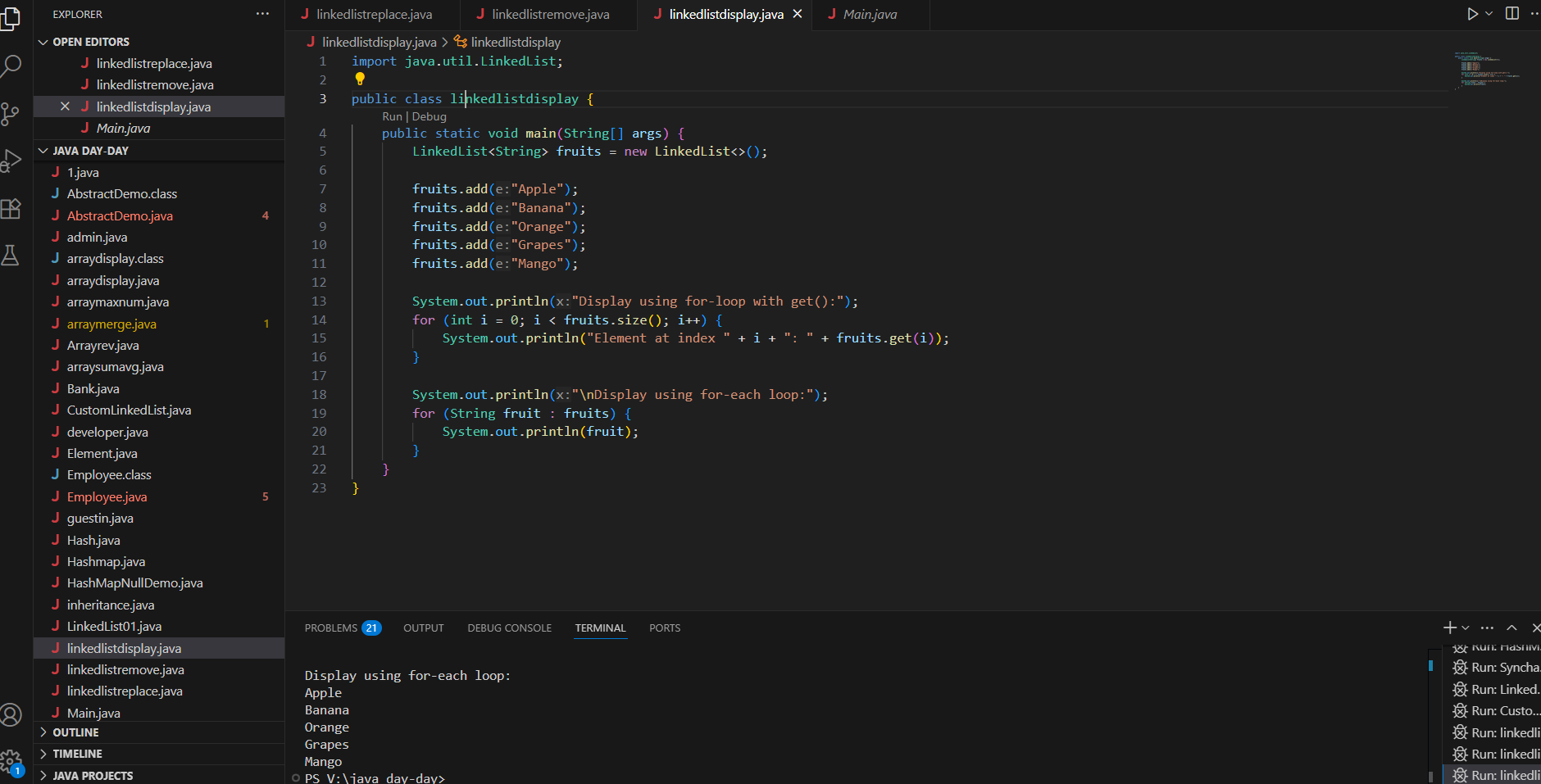
        System.out.println("disp;lay after the Object Removal "+ ll);

    }

}

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();

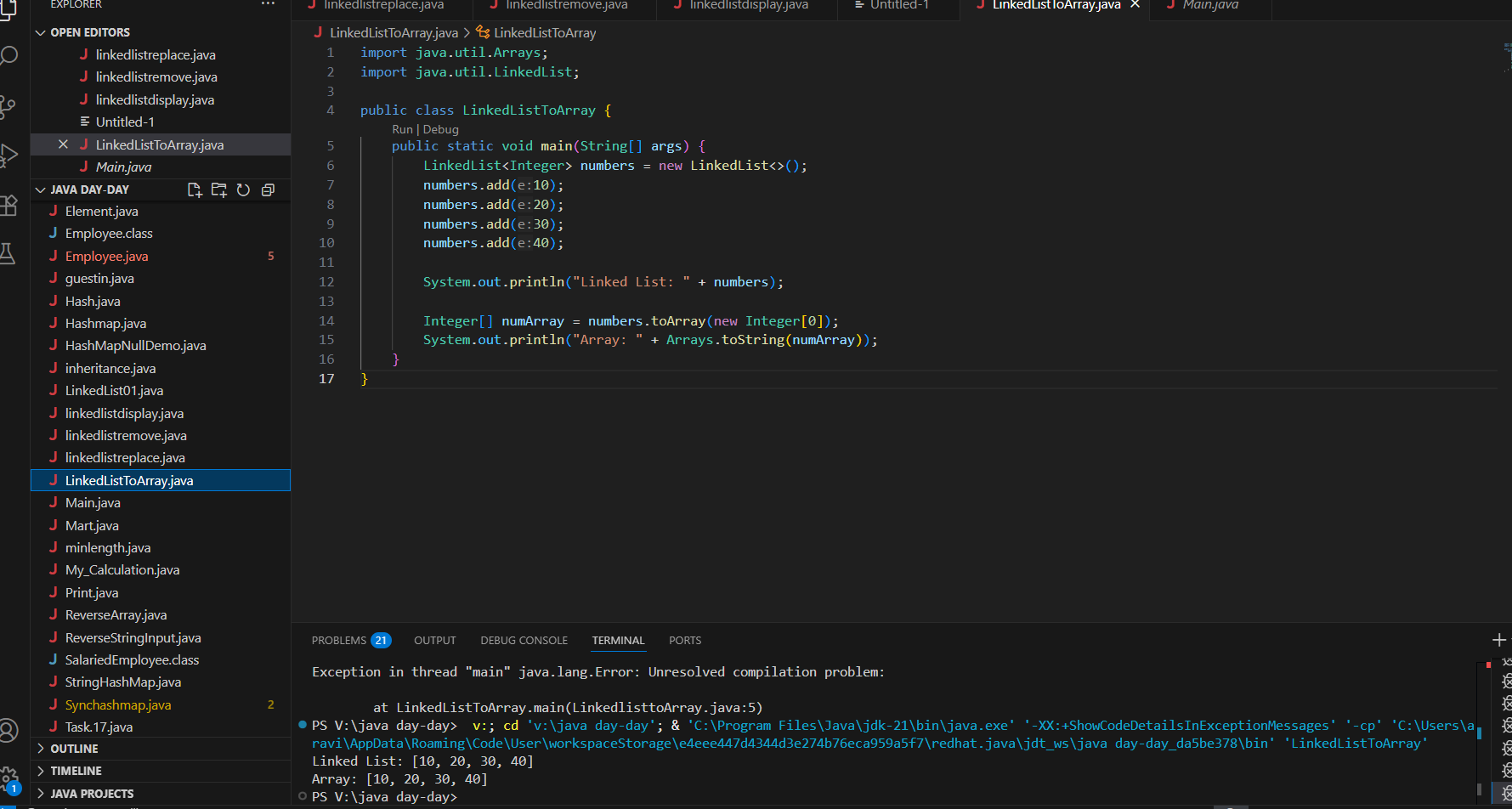
        System.out.print("linked list to array conversion: ");

        for(Object element : a)

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

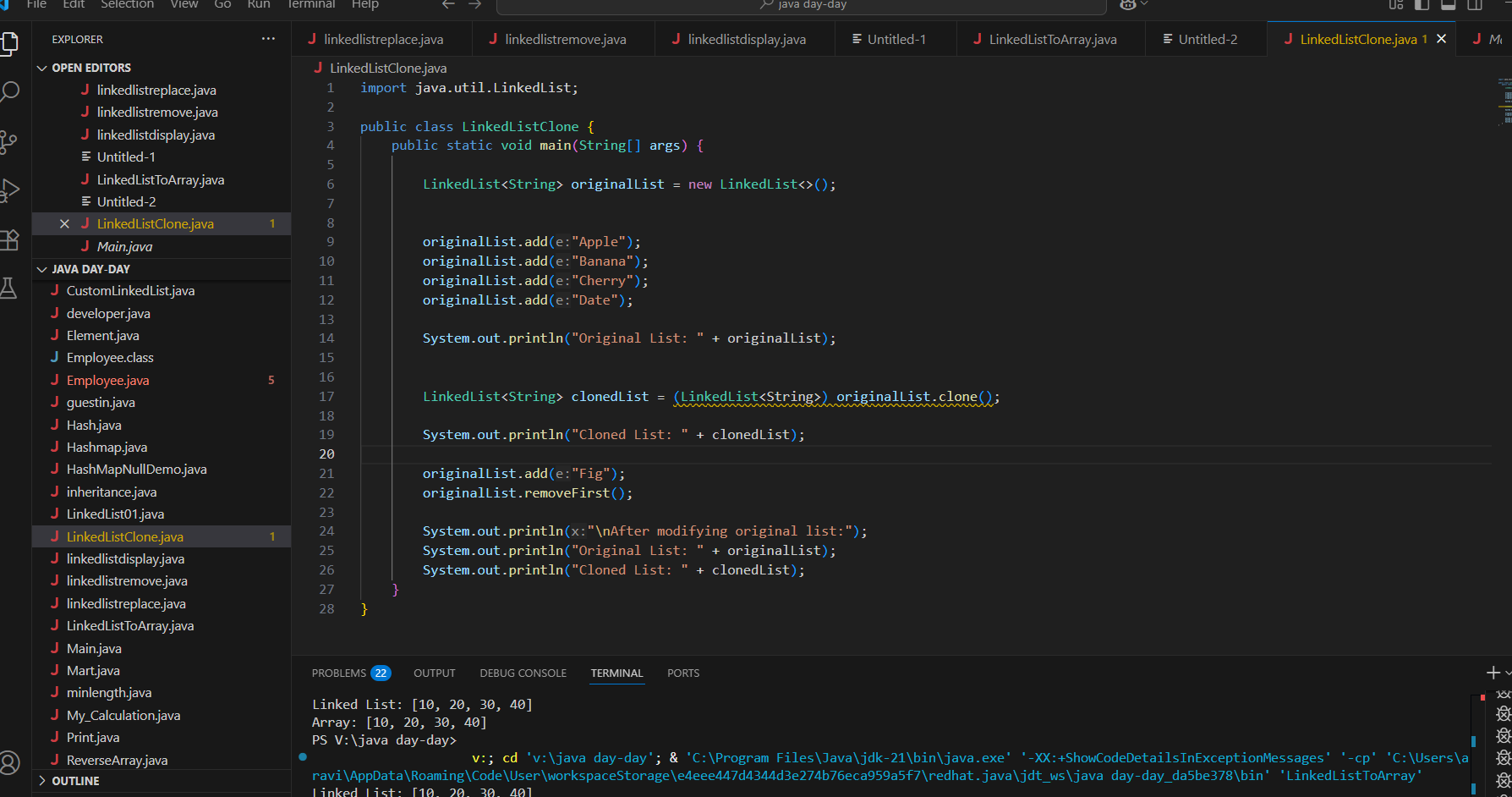
    }

}



Task 9:

Createa alinked list add few items and clone the 1st linked list with the 2nd linked list



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();

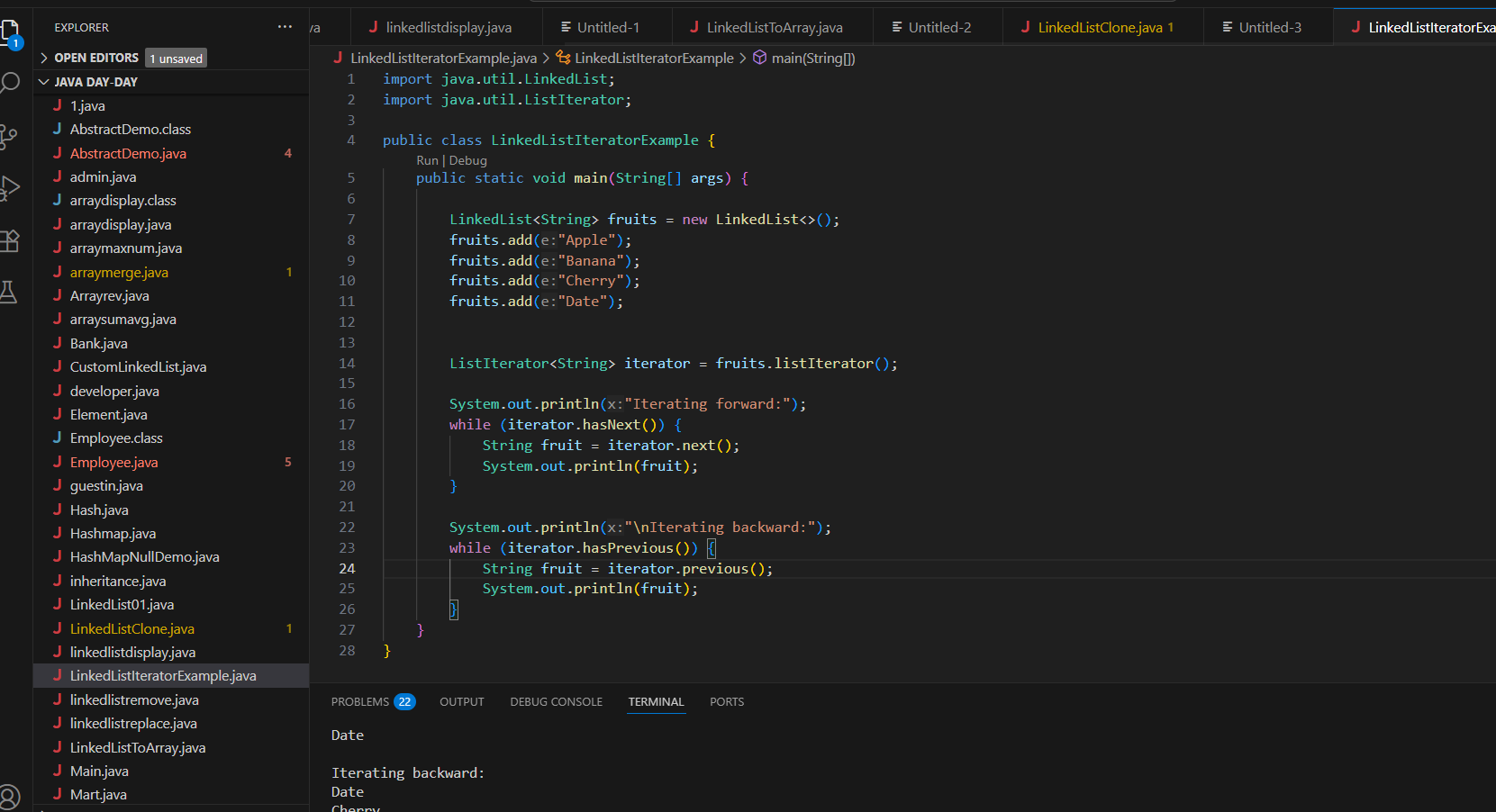
        System.out.println("after cloning ll2 has \n: " + ll2);

    }

}

Task 10:

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



import java.util.LinkedList;

import java.util.ListIterator;

public class Task0010\_DS\_Linkedlist\_ListIterator  {

    public static void main(String args[]) {

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

        llobj.add("Prasunamba");

        llobj.add("Meher");

        llobj.add(".MK");

        ListIterator<String> itobj = llobj.listIterator();

        while (itobj.hasNext()) {

            System.out.print(itobj.next() + " ");

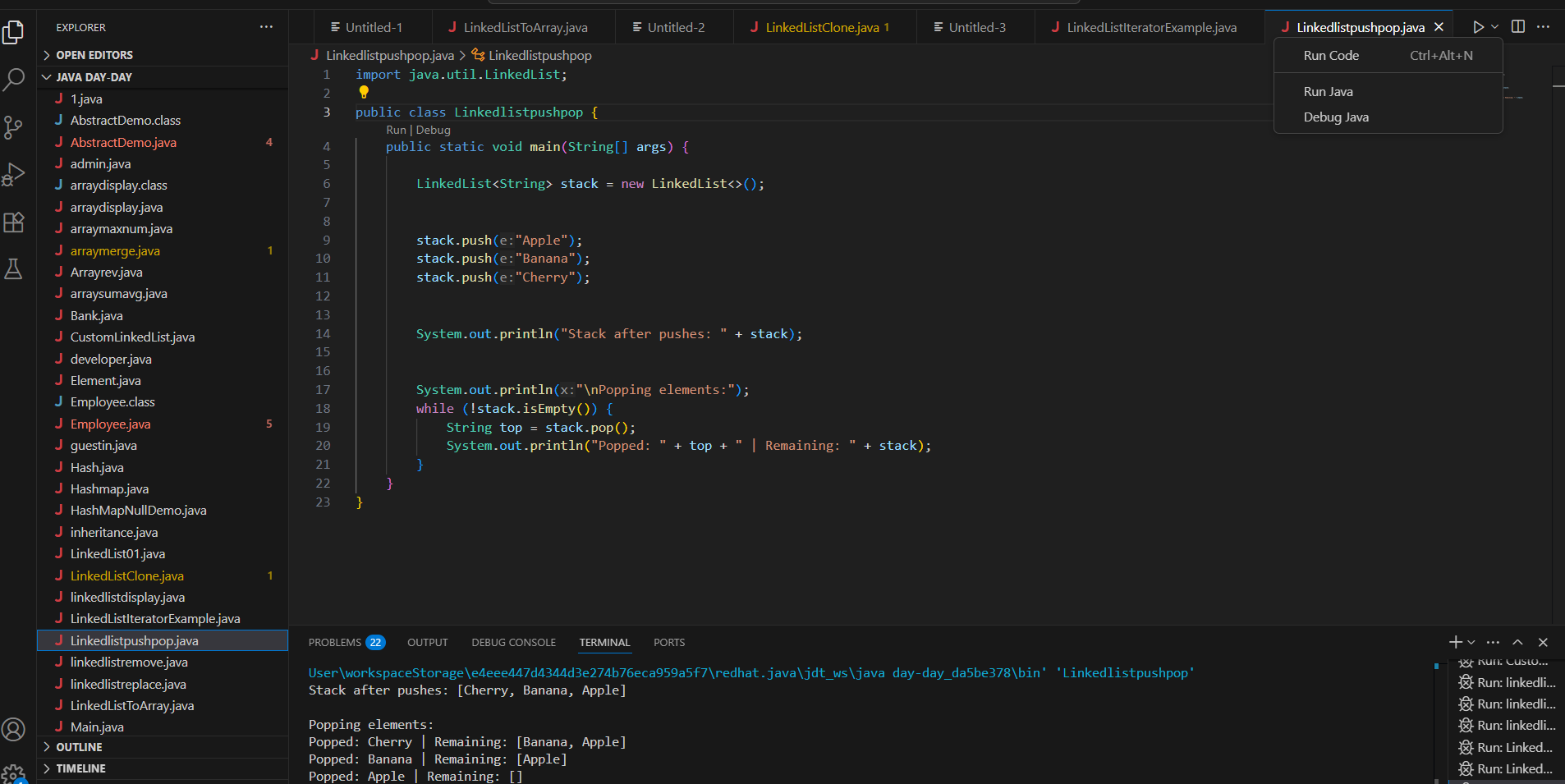
        }

    }

}

Tsk 11:

Create a linked list and use push and pop methods.



Task 12:

Difference between Iterator and splitIterator

1. Parallelism:

- `Iterator`: Strictly sequential.

- `Spliterator`: Enables parallel execution by splitting data.

2. Functionality:

- `Iterator`: Basic traversal + optional removal (`remove()`).

- `Spliterator`: Advanced features like sizing (`estimateSize()`), splitting (`trySplit()`), and bulk traversal (`forEachRemaining()`).

3. Performance:

- `Spliterator` optimizes parallel stream processing (e.g., `stream.parallel()`).

- `Iterator` is lightweight for simple loops.

4. Design Philosophy:

- `Iterator`: Universal but limited.

- `Spliterator`: Engineered for modern multi-core systems.

Task 13:

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

Can you it to sout()... and see ..

16.38 to 16.43

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");

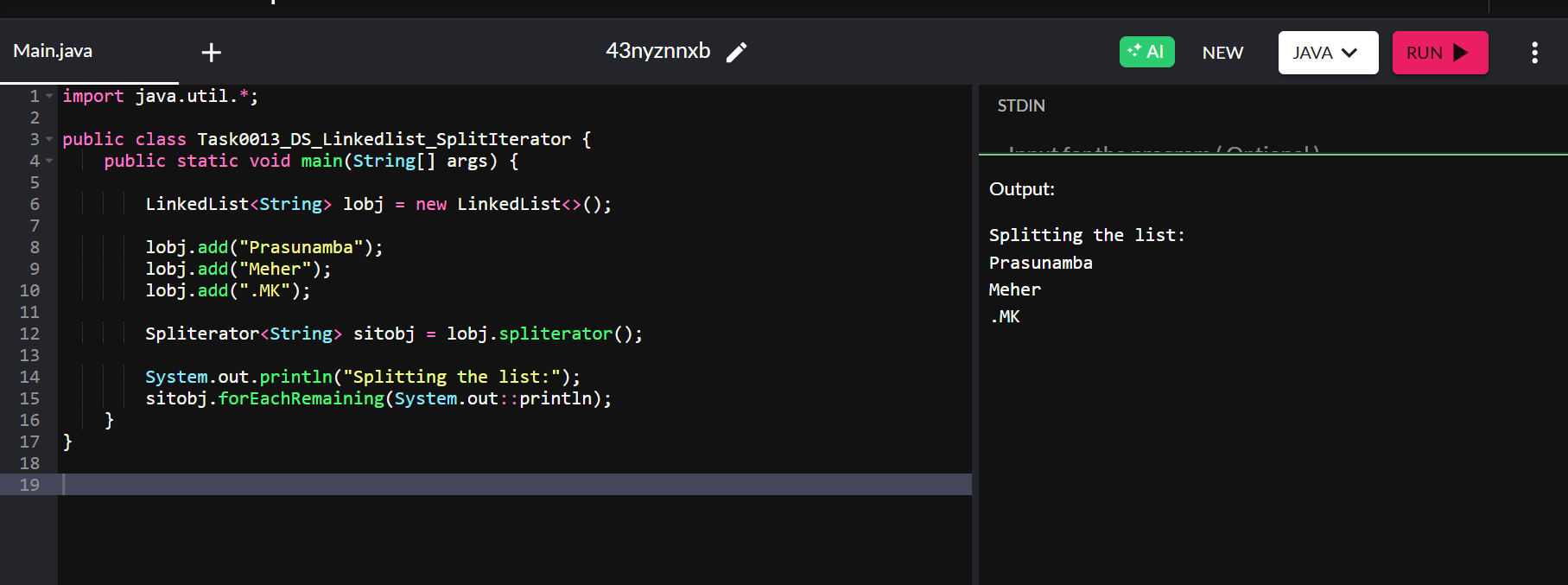
        Spliterator<String> sitobj = lobj.spliterator();

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

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

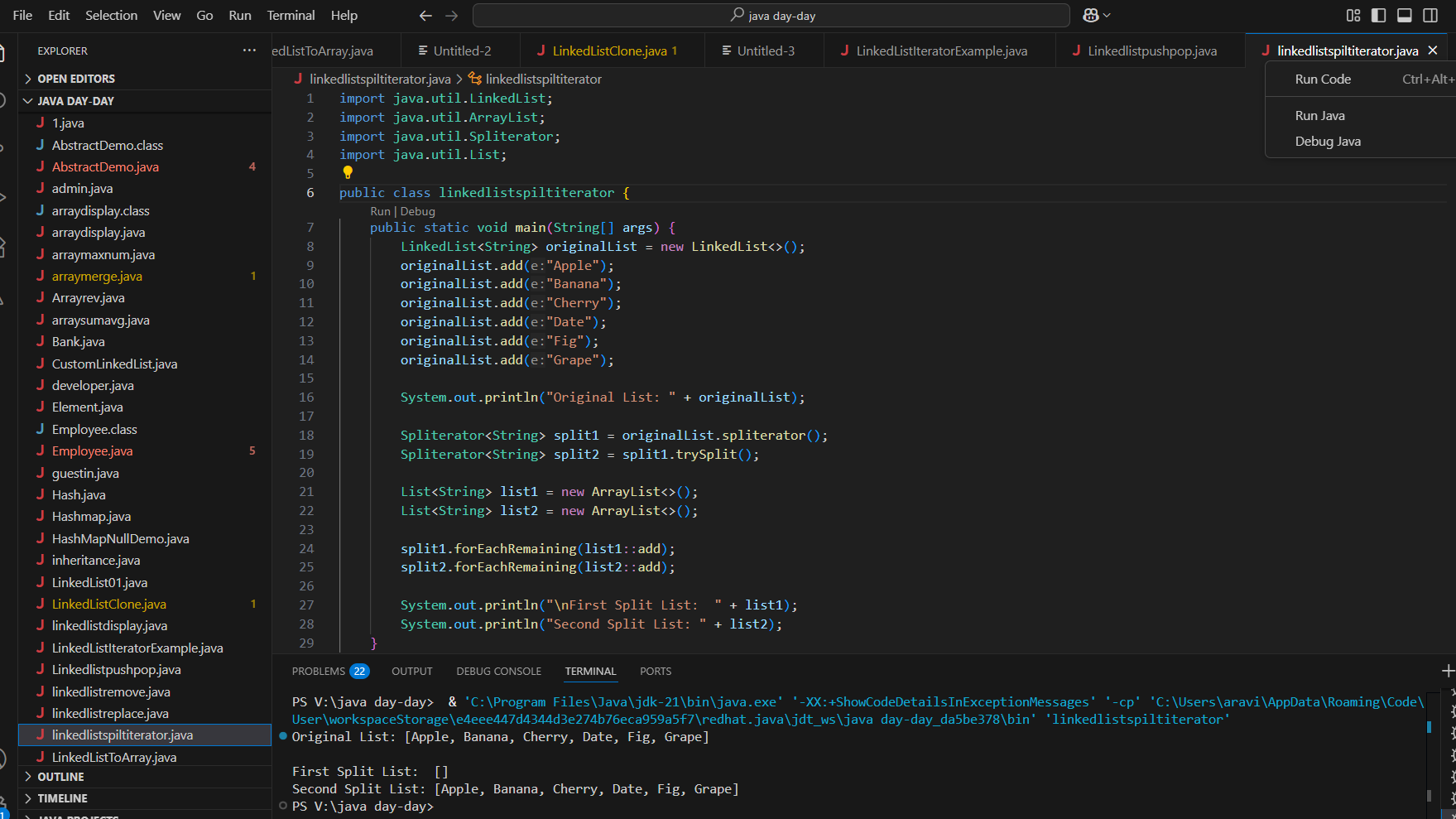
    }

}



Task 14:

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



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); } ) );

  }

}