Pointers:

Pointers are special variables in C that store memory addresses but not values.

#include <stdio.h>

int method1() {

int age;

int \*ptr;

age = 10;

ptr = &age;

printf("Value of age is: %d\n", age);

printf("ptr is pointing to: %d\n", \*ptr);

printf("Address of age is: %p\n", (void\*)&age);

printf("Value of ptr is: %p\n", (void\*)ptr);

printf("Address of ptr itself is: %p\n", (void\*)&ptr);

return 0;

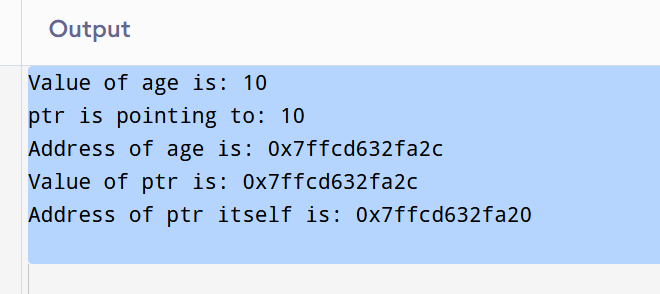
}

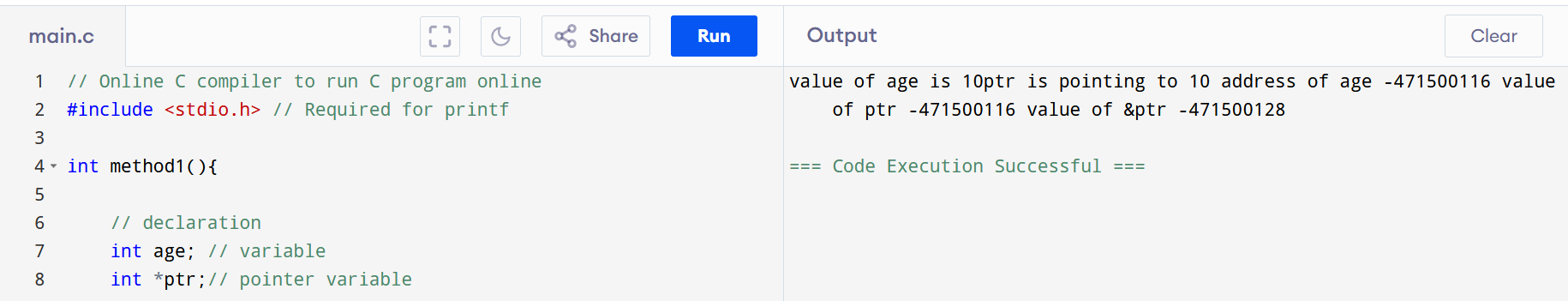
int main() {

method1(); // Call your method1 function here

return 0;

}





Task 2:

#include <bits/stdc++.h>

using namespace std;

// Define a Node class

class Node {

public:

int data;

Node\* next;

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

};

// Class for singly linked list

class LinkedList {

private:

Node\* head;

public:

LinkedList() {

head = nullptr;

}

void insertAtEnd(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

void deleteByValue(int value) {

if (head == nullptr) return;

if (head->data == value) {

Node\* temp = head;

head = head->next;

delete temp;

return;

}

Node\* temp = head;

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

temp = temp->next;

}

if (temp->next) {

Node\* nodeToDelete = temp->next;

temp->next = temp->next->next;

delete nodeToDelete;

}

}

void display() {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << "->";

temp = temp->next;

}

cout << "NULL" << endl;

}

// Destructor to free memory

~LinkedList() {

Node\* temp;

while (head) {

temp = head;

head = head->next;

delete temp;

}

}

};

// Main function

int main() {

LinkedList list; // ✅ Consistent class name

list.insertAtEnd(10);

list.insertAtEnd(20);

list.insertAtEnd(30);

cout << "Linked List: ";

list.display();

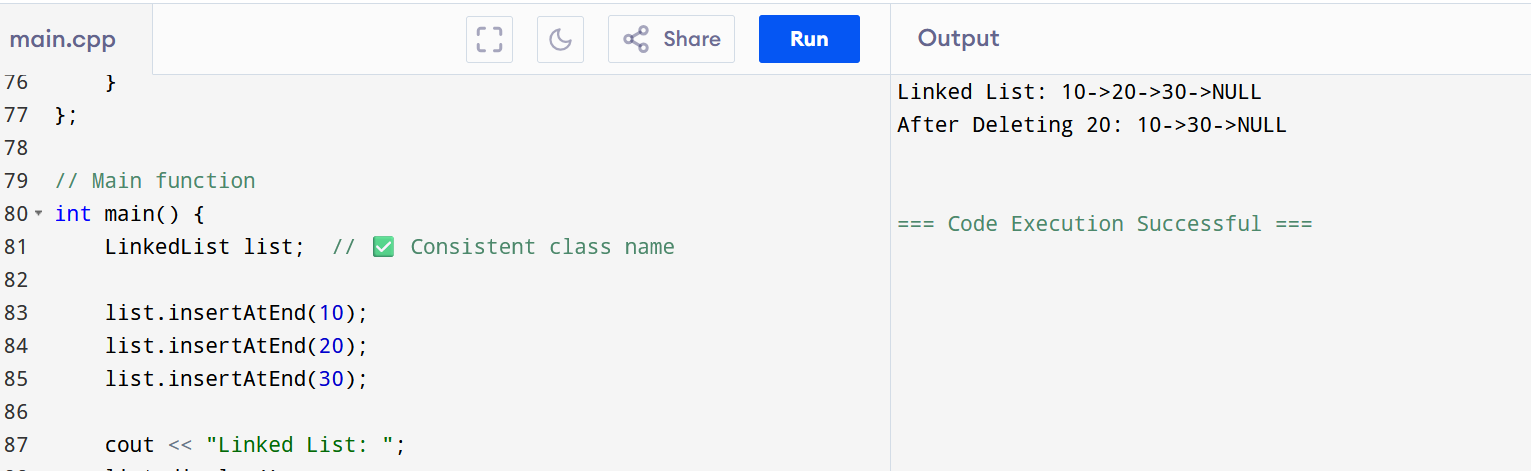
list.deleteByValue(20);

cout << "After Deleting 20: ";

list.display();

return 0;

}



Important notes:

We’re creating node, linkedlist, pointers, there is no pre defined in C++.

In main method, we’re doing 6 operations:

3 INSERT

TRAVERSAL (DISPLAY)

DELETING A NODE

TRAVERSAL

public class LinkedList<T> {

Node<T> head;

// Inner generic Node class

static class Node<T> {

T data;

Node<T> next;

public Node(T value) {

data = value;

next = null;

}

}

// Insert at end

public void insertAtEnd(T value) {

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

if (head == null) {

head = newNode;

} else {

Node<T> temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

// Display the list

public void display() {

Node<T> temp = head;

while (temp != null) {

System.out.print(temp.data + " -> ");

temp = temp.next;

}

System.out.println("NULL");

}

// Delete by value

public void deleteByValue(T value) {

if (head == null) return;

if (head.data.equals(value)) {

head = head.next;

return;

}

Node<T> temp = head;

while (temp.next != null && !temp.next.data.equals(value)) {

temp = temp.next;

}

if (temp.next != null) {

temp.next = temp.next.next;

}

}

// Main to test

public static void main(String[] args) {

// Create LinkedList of Strings

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

stringList.insertAtEnd("Good");

stringList.insertAtEnd("Luck");

stringList.display();

// Create LinkedList of Integers

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

intList.insertAtEnd(10);

intList.insertAtEnd(20);

intList.insertAtEnd(30);

intList.display();

// Create LinkedList of Doubles

LinkedList<Double> floatList = new LinkedList<>();

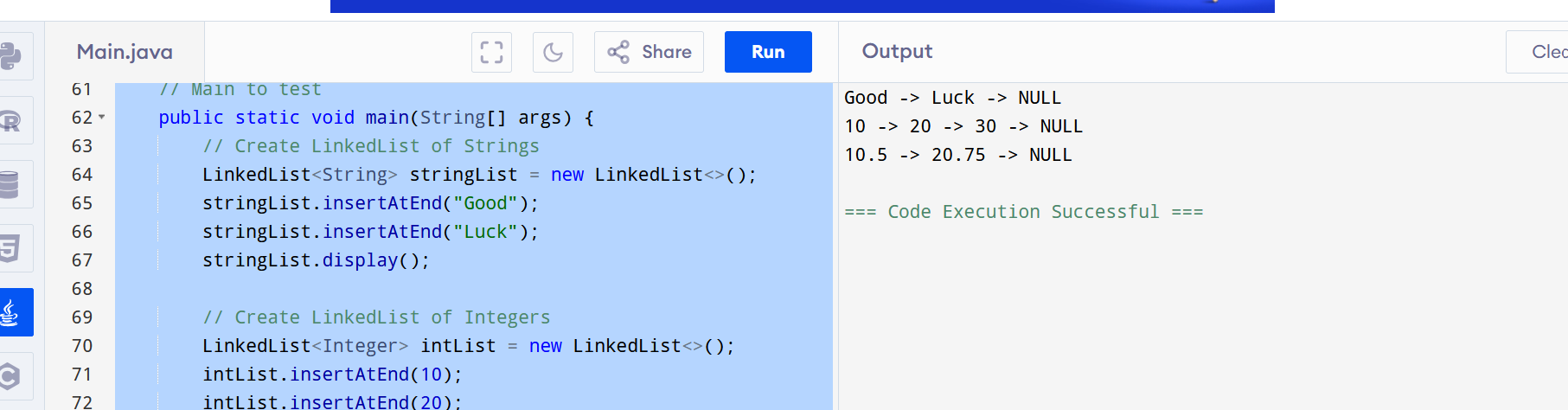
floatList.insertAtEnd(10.5);

floatList.insertAtEnd(20.75);

floatList.display();

}

}



Task5:

List down all the methods of Linked list

insertAtBeginning(int value) Adds a new node at the **start** of the list

insertAtEnd(int value) Adds a new node at the **end** of the list

insertAtPosition(int index, int value) Adds a new node at a specific **position** (0-based)

deleteFromBeginning() Deletes the **first** node

deleteFromEnd() Deletes the **last** node

deleteByValue(int value) Deletes the **first node** with the given value

deleteAtPosition(int index) Deletes the node at a given index

search(int value) Searches for a value and returns its **position** or -1 if not found

isEmpty() Returns true if the list is empty

getSize() / length() Returns the total number of nodes

getMiddle() Returns the **middle node's value**

reverse() Reverses the entire list (optional advanced method)

display() Prints all the node values in order

toArray() Converts the list into an array (for testing or visualization)

Predefined methods

| **Method** | **What it does** |
| --- | --- |
| add() | Add to end |
| addFirst() / addLast() | Add at beginning / end |
| add(index, value) | Add at specific position |
| removeFirst() / removeLast() | Remove from beginning / end |
| remove(value) | Remove specific value |
| contains(value) | Check if value exists |
| size() | Total elements |
| get(index) | Get value at position |

Task6:

import java.util.LinkedList;

public class task6{

public static void main(String[] args){

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

list.add(10);

list.add(20);

list.add(90);

System.out.println("LinkedList: "+ list);

int first=list.getFirst();

System.out.println("First element: "+ first);

int last=list.getLast();

System.out.println("Last element: "+ last);

}

}



Task 7: remove first and last elements and display all the elements.

import java.util.LinkedList;

public class task7{

public static void main(String[] args){

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

list.add(10);

list.add(20);

list.add(90);

System.out.println("LinkedList: "+ list);

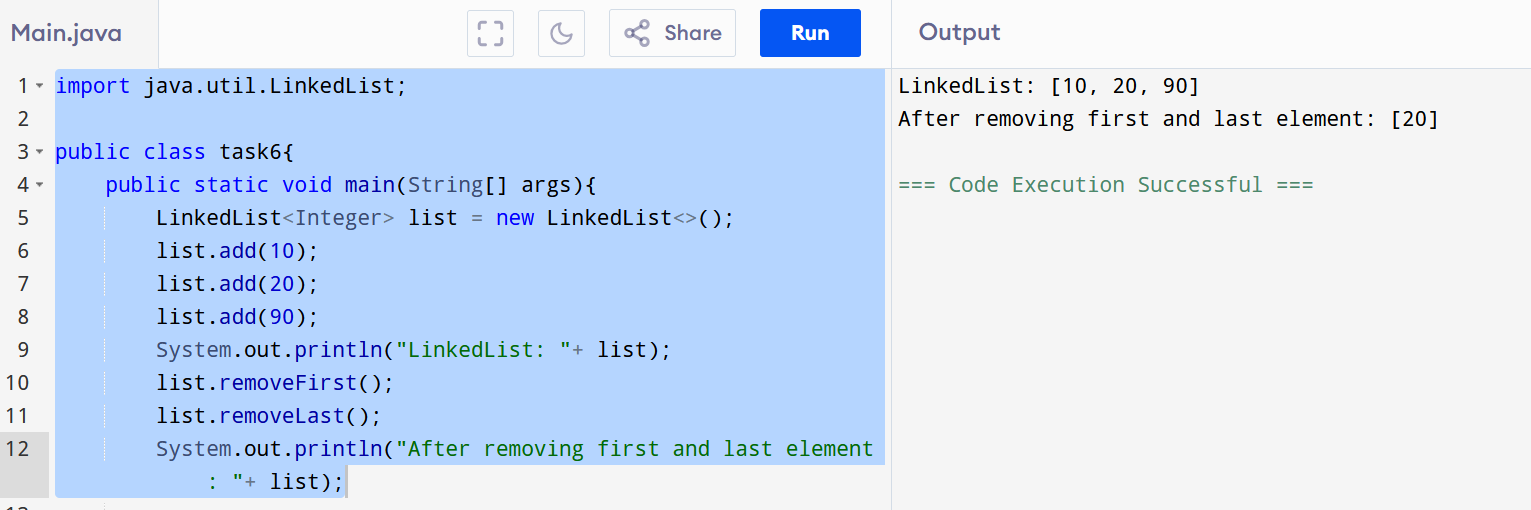
list.removeFirst();

list.removeLast();

System.out.println("After removing first and last element: "+ list);

}

}



Task8: in the list update the 1st element to a new value Hint: use set(1, "new value");

import java.util.LinkedList;

public class task8{

public static void main(String[] args){

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

list.add(10);

list.add(20);

list.add(90);

System.out.println("LinkedList: "+ list);

list.set(0, 100);

list.set(1,101);

System.out.println("After updating the element: "+ list);

}

}



Task 9: display the list twice 1..... with get method in for loop and 2 ... for each loop

import java.util.LinkedList;

public class task9 {

public static void main(String[] args) {

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

list.add(10);

list.add(20);

list.add(30);

System.out.print("Using for loop with get(): ");

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

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

}

System.out.print("Using for-each loop: ");

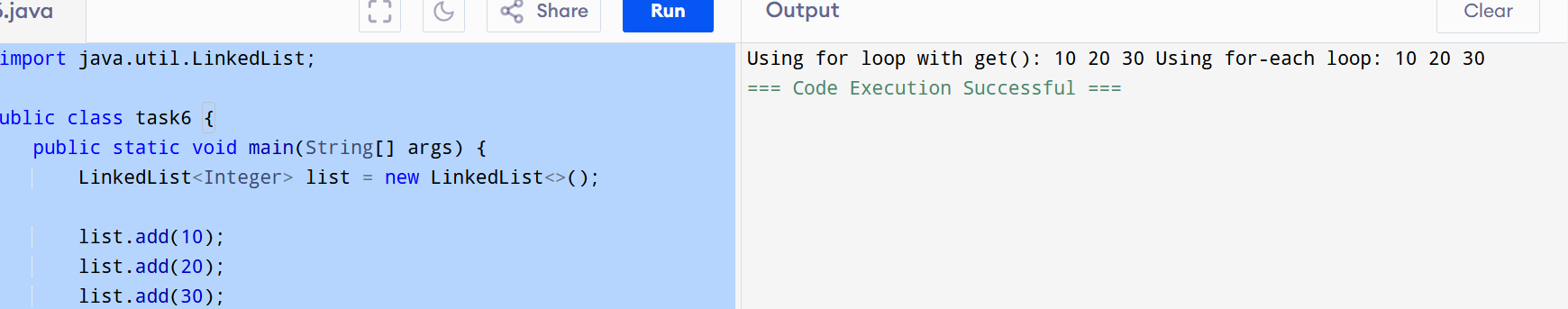
for (int num : list) {

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

}

}

}



Task 10: display the elements of the linked list with out loops

import java.util.LinkedList;

public class task10 {

public static void main(String[] args) {

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

list.add(10);

list.add(20);

list.add(30);

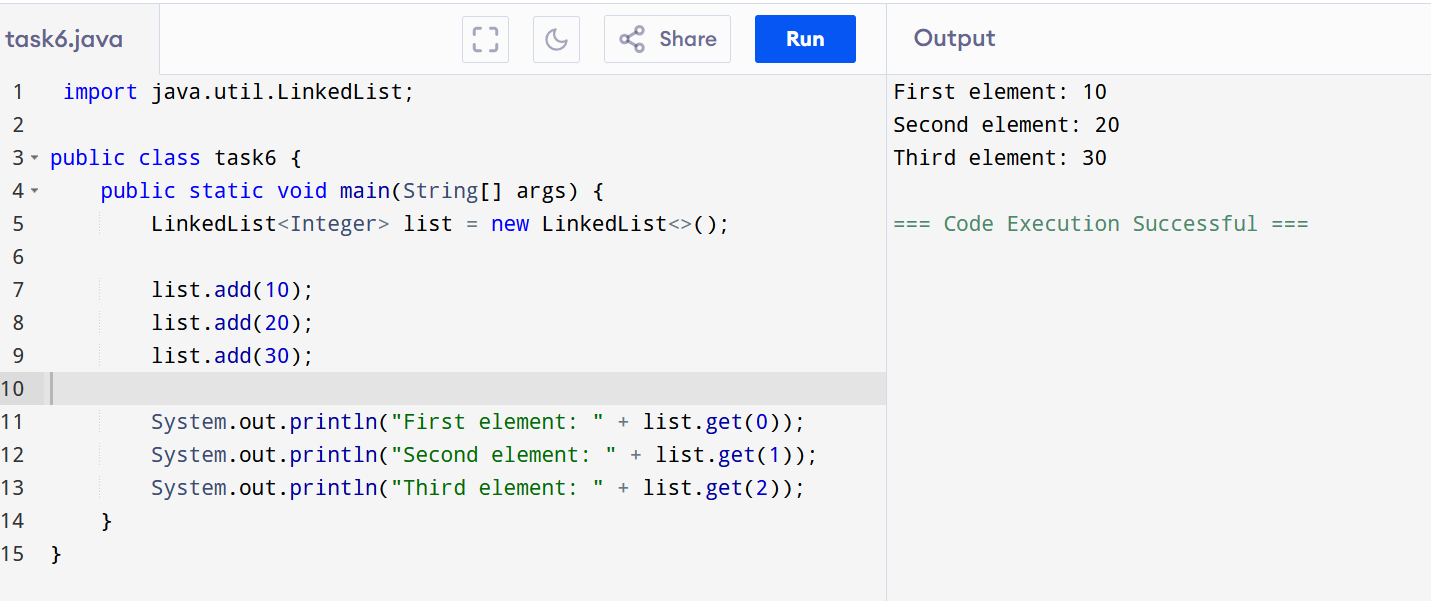
System.out.println("First element: " + list.get(0));

System.out.println("Second element: " + list.get(1));

System.out.println("Third element: " + list.get(2));

}

}



task 11: convert the linked list to an array and display Hint : Object[] a = llobj.toArray();

import java.util.LinkedList;

public class task11 {

public static void main(String[] args) {

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

list.add(10);

list.add(20);

list.add(30);

Object[] arr = list.toArray();

System.out.println("Accessing manually: " + arr[0] + ", " + arr[1] + ", " + arr[2]);

System.out.print("Array elements: ");

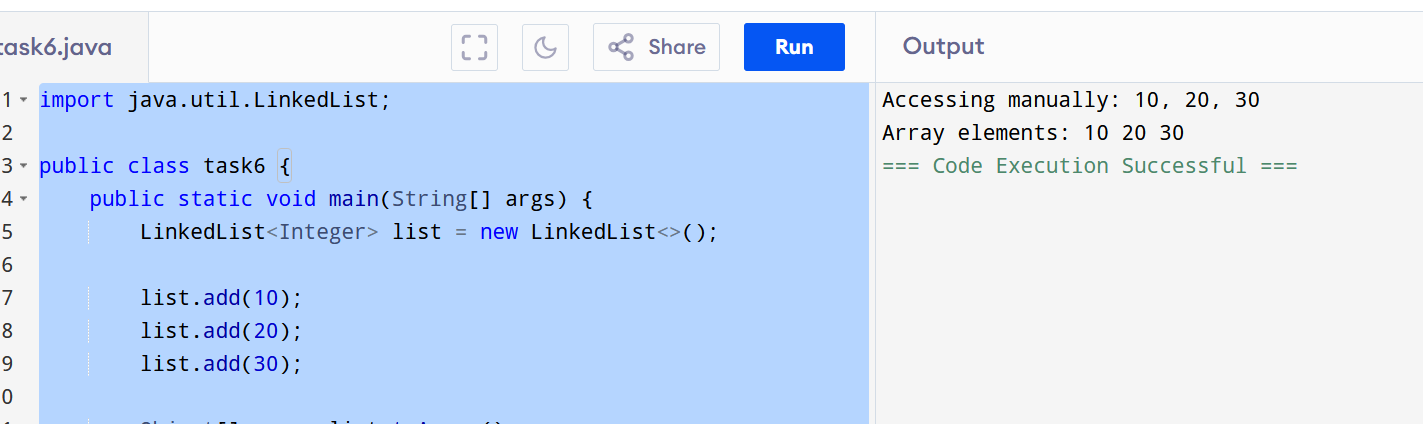
for (Object val : arr) {

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

}

}

}



Task 12: clone the linked list to check if its getting cloned?

import java.util.LinkedList;

public class task6 {

public static void main(String[] args) {

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

originalList.add(10);

originalList.add(20);

originalList.add(30);

LinkedList<Integer> clonedList = (LinkedList<Integer>) originalList.clone();

System.out.println("Original List: " + originalList);

System.out.println("Cloned List: " + clonedList);

originalList.add(999);

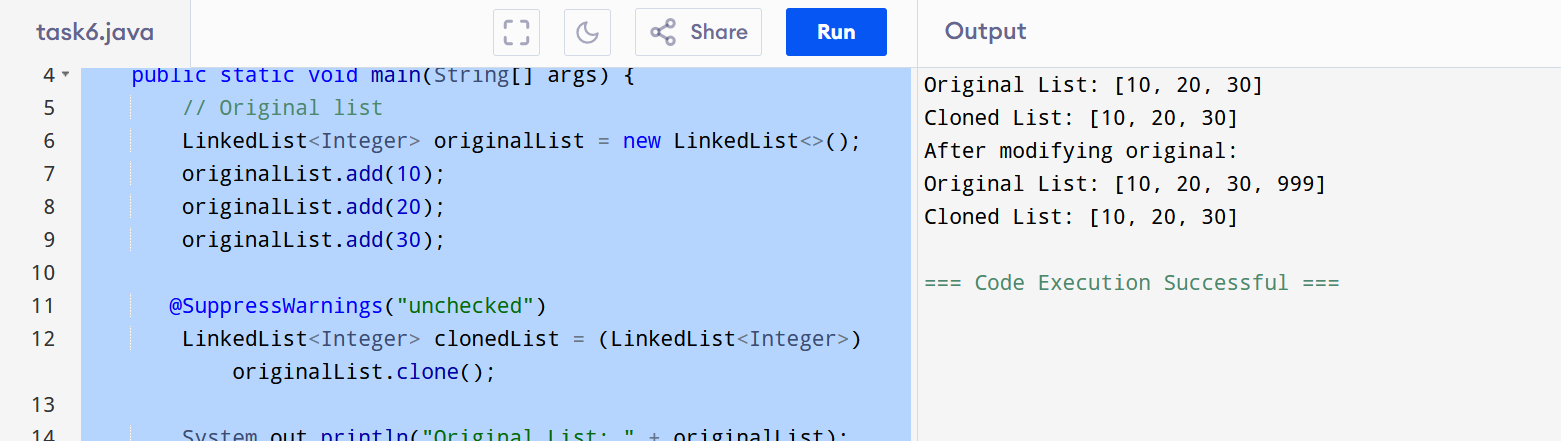
System.out.println("After modifying original:");

System.out.println("Original List: " + originalList);

System.out.println("Cloned List: " + clonedList);

}

}



Task 13:

Use pop and push methods on linked list..

import java.util.LinkedList;

public class task13 {

public static void main(String[] args) {

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

list.push(10);

list.push(20);

list.push(30);

list.push(40);

list.push(90);

System.out.println("After pushing the elements: " + list);

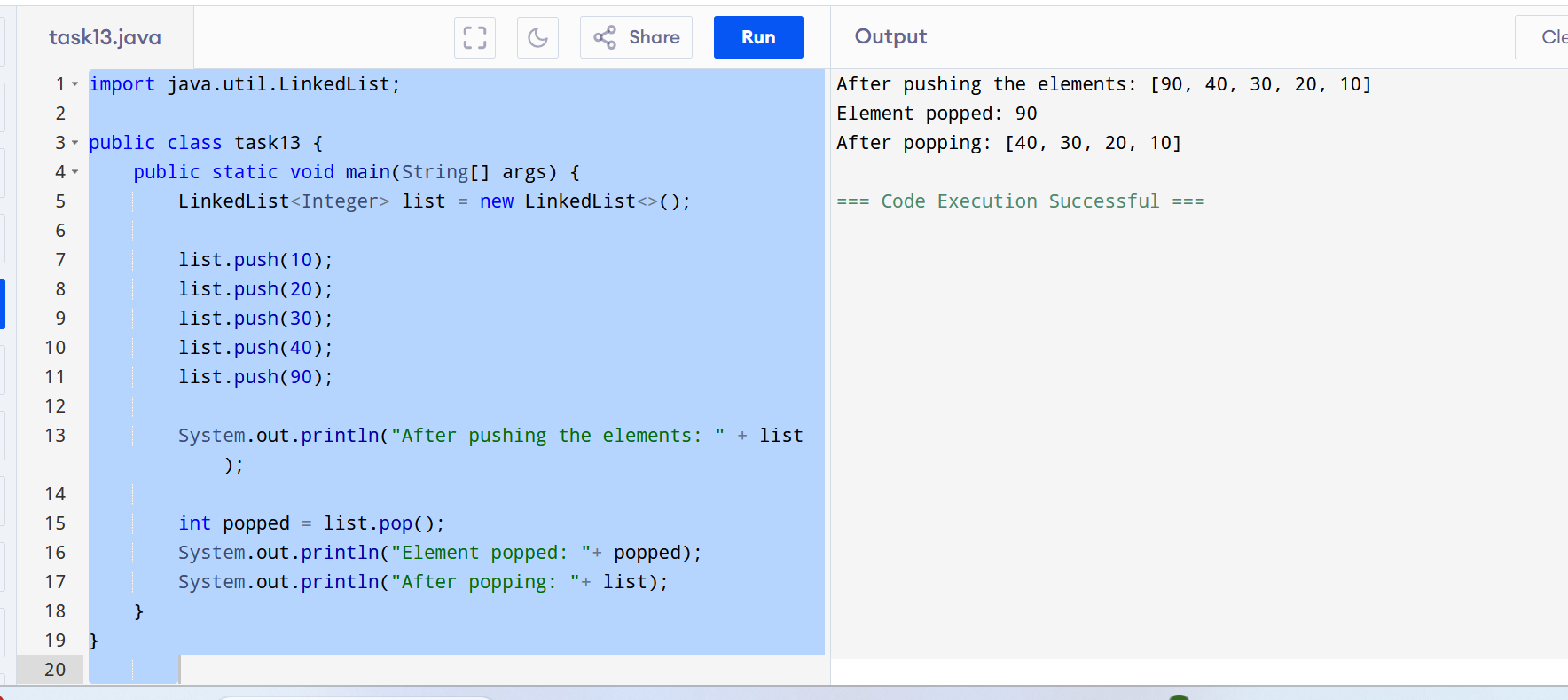
int popped = list.pop();

System.out.println("Element popped: "+ popped);

System.out.println("After popping: "+ list);

}

}



Task 14: Splititerator  
It's an advanced iterator that can **traverse** and **split** collections — often used with **Streams** and **parallelism**.

import java.util.\*;

public class Task0014\_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();

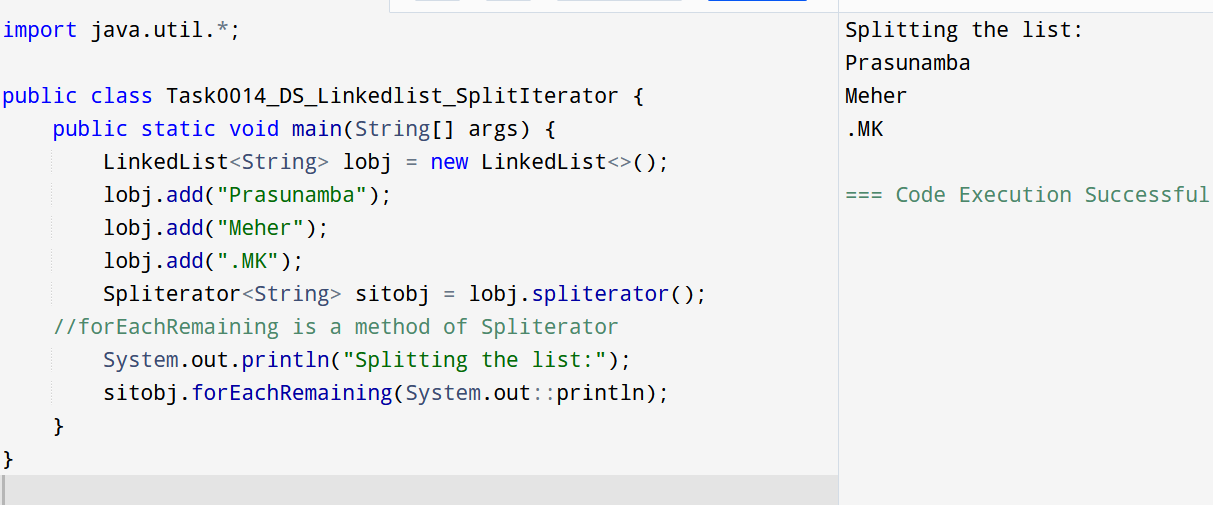
//forEachRemaining is a method of Spliterator

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

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

    }

}



Task 15:

tryAdvance()

import java.util.LinkedList;

import java.util.Spliterator;

public class Task0015\_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");

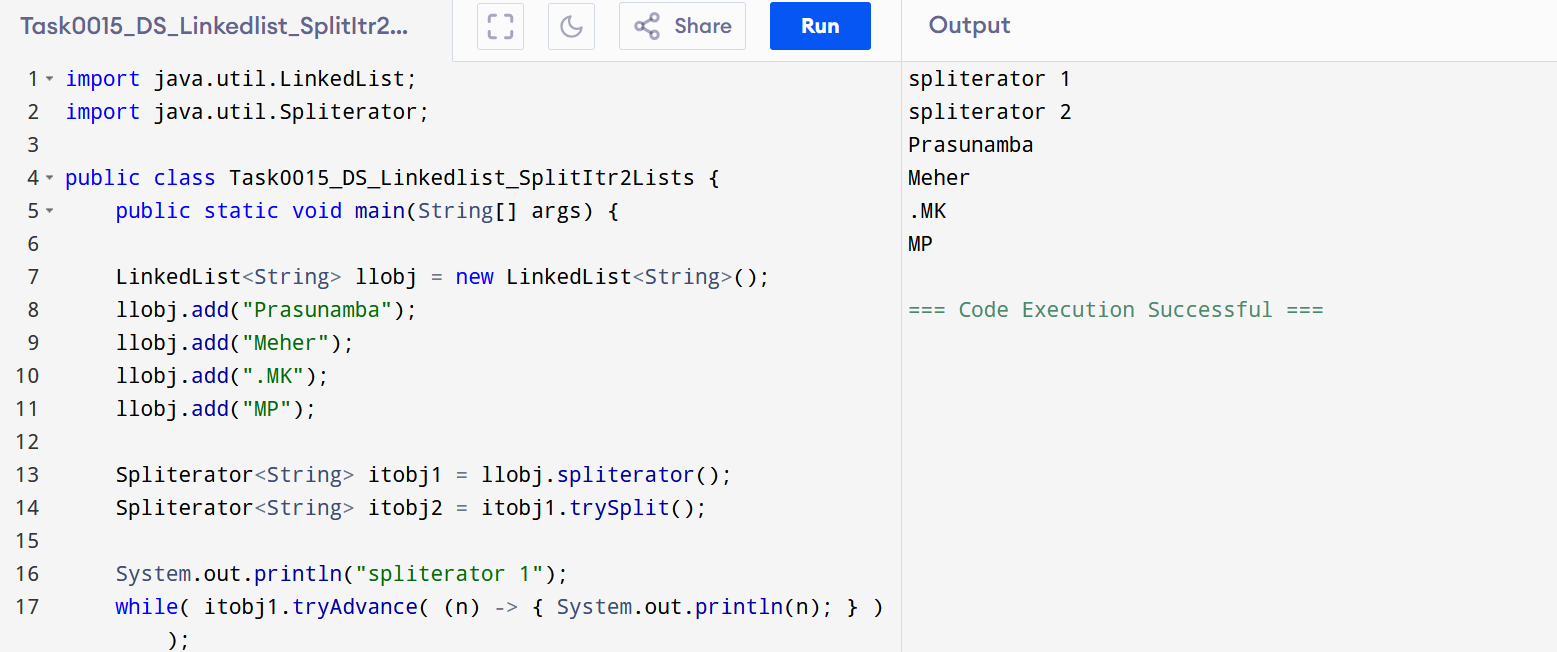
    while( itobj1.tryAdvance( (n) -> { System.out.println(n); } ) );

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

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

  }

}



public class DoublyLinkedListDemo {

static class Node {

int data;

Node prev;

Node next;

Node(int data) {

this.data = data;

this.prev = null;

this.next = null;

}

}

// Doubly Linked List logic

static class DoublyLinkedList {

Node head;

// Inserting node at the end

void insertAtEnd(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

newNode.prev = temp;

}

// Display list from head

void displayForward() {

Node temp = head;

System.out.print("Forward: ");

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

// Display list from tail

void displayBackward() {

Node temp = head;

// Go to the last node

while (temp != null && temp.next != null) {

temp = temp.next;

}

System.out.print("Backward: ");

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.prev;

}

System.out.println();

}

}

// Main method

public static void main(String[] args) {

DoublyLinkedList dll = new DoublyLinkedList();

dll.insertAtEnd(10);

dll.insertAtEnd(20);

dll.insertAtEnd(30);

dll.displayForward(); // Output: Forward: 10 20 30

dll.displayBackward(); // Output: Backward: 30 20 10

}

}

