Task 01

#include<stdio.h>  //-- c lang

#include<iostream.h> // -- c++

int method1(){

declaration

int age; // variable

int \*ptr;// pointer variable

// assigning

age = 10;

ptr = &age;

printf("value of age is %d", age); // 10

printf("ptr is pointing to %d ", \*ptr); // 10

printf("address of age %d ", &age);// 145245

printf("value of ptr %d ", ptr);//same as above 145245

printf("value of &ptr %d ",&ptr); // ptr address --- 454578

if c++

cout<<"value of age "<<age;

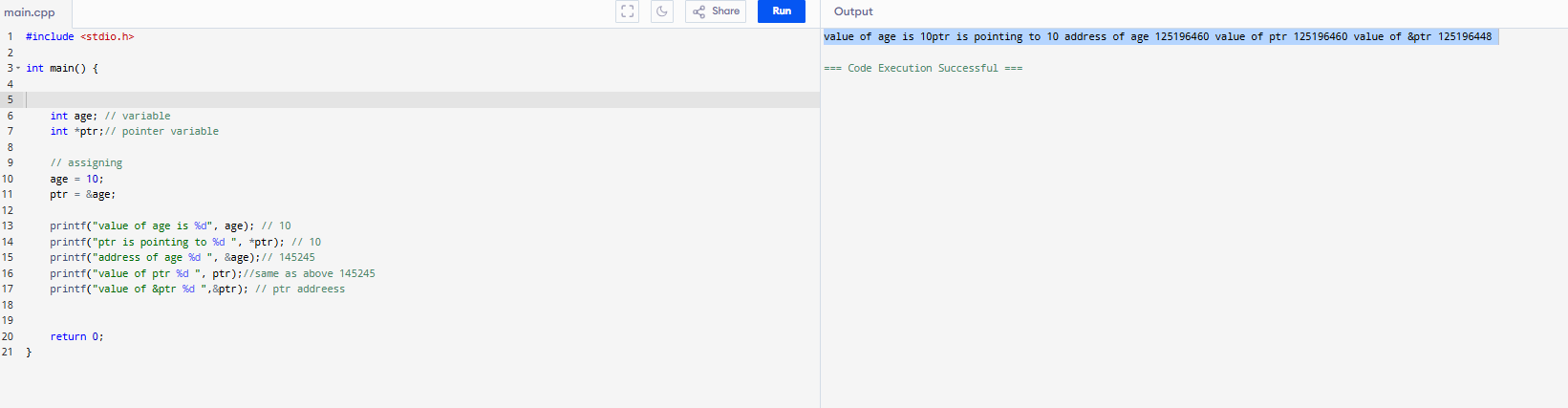
cout<<"ptr is pointing to"<<\*ptr;

cout<<"address of age "<<&age;

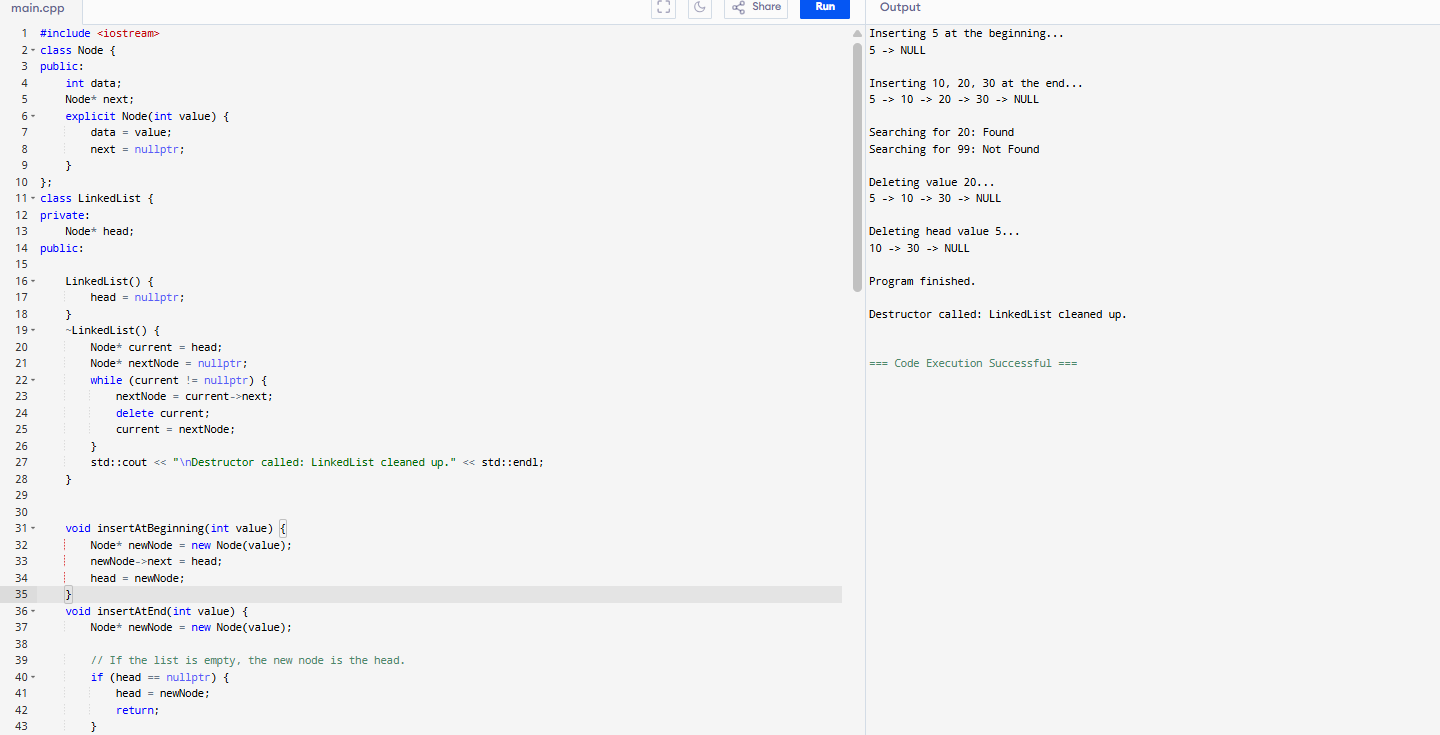
cout<<"value of ptr "<<ptr;

cout<<"value of &ptr "<<&ptr;

}



Task 2:



Task 3:

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

public class Task3 {

private Node head;

public Task3() {

this.head = null;

}

public void insertAtEnd(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

} else {

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

public void deleteByValue(int value) {

if (head == null) {

return;

}

if (head.data == value) {

head = head.next;

return;

}

Node temp = head;

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

temp = temp.next;

}

if (temp.next != null) {

Node nodeToDelete = temp.next;

temp.next = temp.next.next;

// The nodeToDelete will be garbage collected

}

}

public void display() {

Node temp = head;

while (temp != null) {

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

temp = temp.next;

}

System.out.println("NULL");

}

public static void main(String[] args) {

Task3 list = new Task3();

list.insertAtEnd(10);

list.insertAtEnd(20);

list.insertAtEnd(30);

System.out.print("Linked List: ");

list.display();

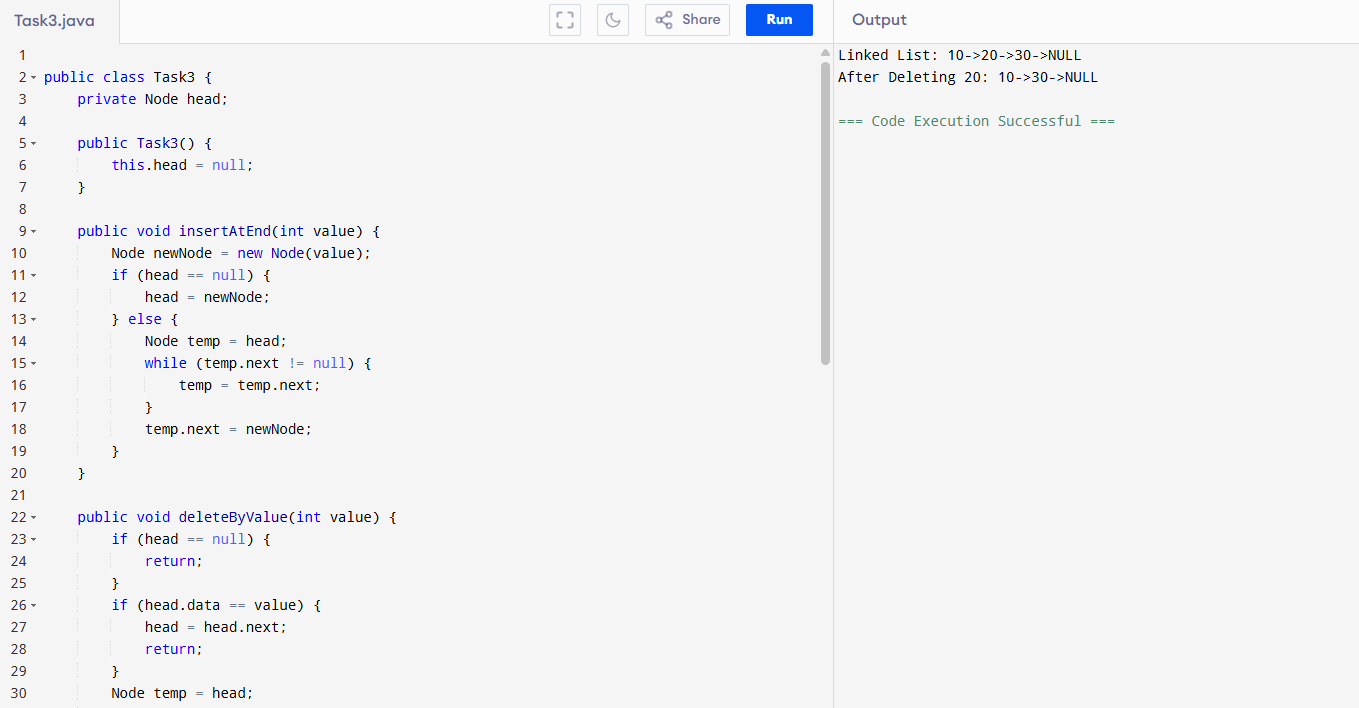
list.deleteByValue(20);

System.out.print("After Deleting 20: ");

list.display();

}

}



Task 4:

import java.util.NoSuchElementException;

public class MyFlexibleList<T> {

private static class Node<T> {

T data;

Node<T> next;

public Node(T data) {

this.data = data;

this.next = null;

}

}

private Node<T> head;

private int size;

public MyFlexibleList() {

this.head = null;

this.size = 0;

}

public void add(T data) {

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

if (head == null) {

head = newNode;

} else {

Node<T> walker = head;

while (walker.next != null) {

walker = walker.next;

}

walker.next = newNode;

}

size++;

}

public void remove(int index) {

checkBounds(index);

if (index == 0) {

head = head.next;

} else {

Node<T> walker = head;

for (int i = 0; i < index - 1; i++) {

walker = walker.next;

}

walker.next = walker.next.next;

}

size--;

}

public void display() {

if (head == null) {

System.out.println("List is empty.");

return;

}

Node<T> walker = head;

System.out.print("LIST: [");

while (walker != null) {

System.out.print(walker.data);

if (walker.next != null) {

System.out.print(" -> ");

}

walker = walker.next;

}

System.out.println("]");

}

public int getSize() {

return this.size;

}

private void checkBounds(int index) {

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

throw new IndexOutOfBoundsException("Cannot access index " + index + ". List size is " + size + ".");

}

}

public static void main(String[] args) {

MyFlexibleList<String> fruitList = new MyFlexibleList<>();

System.out.println("--- Adding Elements ---");

fruitList.add("Apple");

fruitList.add("Banana");

fruitList.add("Cherry");

fruitList.display();

System.out.println("Current size: " + fruitList.getSize());

System.out.println("\n--- Removing Element at index 1 (Banana) ---");

fruitList.remove(1);

fruitList.display();

System.out.println("Size after removal: " + fruitList.getSize());

System.out.println("\n--- Testing Index Out Of Bounds ---");

try {

System.out.println("Attempting to remove element at index 10 (which is invalid)...");

fruitList.remove(10);

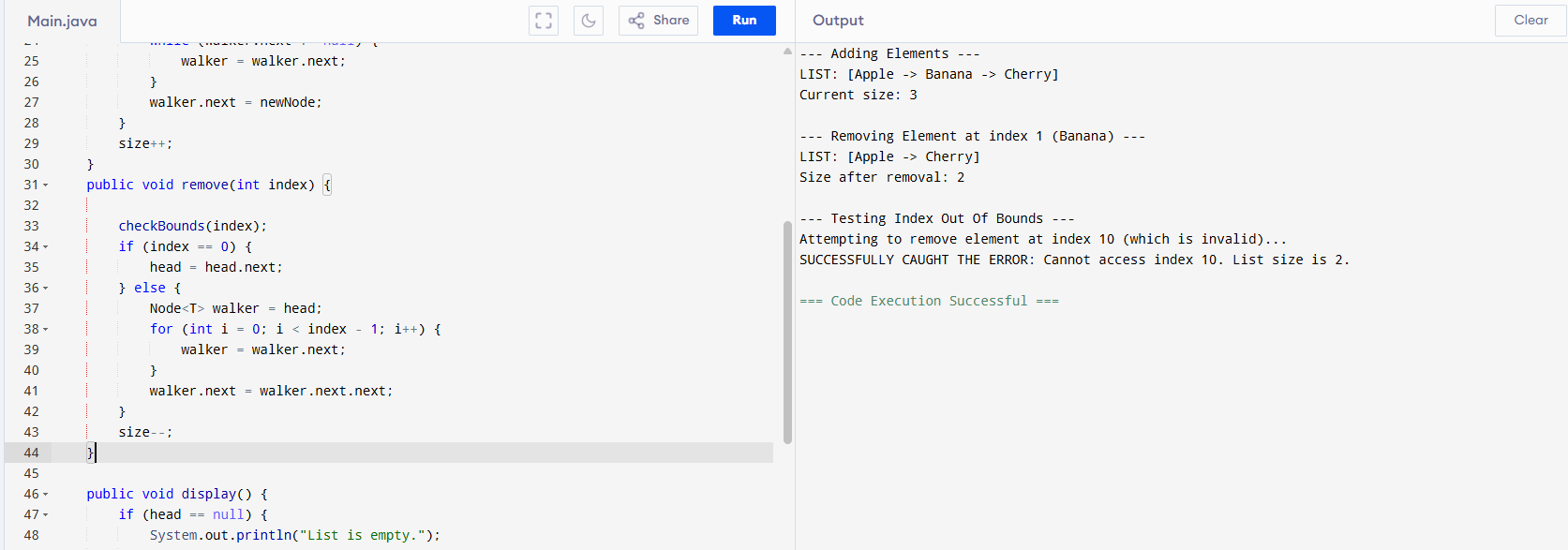
} catch (IndexOutOfBoundsException e) {

System.err.println("SUCCESSFULLY CAUGHT THE ERROR: " + e.getMessage());

}

}

}



Task 5:

Java's `LinkedList` is much more powerful than our simple one because it's a \*\*Doubly-Linked List\*\* (each node knows about the `next` and `previous` node) and it also implements the `List`, `Queue`, and `Deque` (Double-Ended Queue) interfaces. This means it has a rich set of methods for many different use cases.

the most common and important methods, grouped by their function.

1. Methods for Adding Elements

These methods are for inserting new data into the list.

`boolean add(E element)`

`void add(int index, E element)`

`void addFirst(E element)`

`void addLast(E element)`

`boolean addAll(Collection c)`

---

2. Methods for Removing Elements

These methods are for taking data out of the list.

E remove(int index) ,boolean remove(Object o), E remove(), E removeFirst(), Same as `remove()`. ,E removeLast(), void clear()

---

3. Methods for Getting/Examining Elements (Without Removing)

These methods let you look at the data in the list without changing the list.

E get(int index)

E getFirst()

E getLast()

int indexOf(Object o)

int lastIndexOf(Object o)

boolean contains(Object o)

---

4. Utility/Helper Methods

These methods give you information about the list itself.

int size()

boolean isEmpty()

Object[] toArray()

5. Special Methods for Queue & Stack Behavior:Because `LinkedList` can act like a queue or a stack, it has special methods for these behaviors. This is what makes it so flexible!

As a Queue (FIFO: First-In, First-Out)

Think of a lineup at a checkout counter.

offer(E e)

poll()

peek()

As a Stack (LIFO: Last-In, First-Out)

Think of a stack of plates.

push(E e)

pop()

peek()

Task6to10:

import java.util.LinkedList;

import java.util.\*;

public class Task6to10 {

public static void main(String[] args) {

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

fruits.add("Apple ");

fruits.add("Banana ");

fruits.add("Avocado ");

fruits.add("Pear ");

fruits.add("Orange ");

System.out.println("\nElements in linked List: ");

for( String fruit:fruits){

System.out.print(fruit);

}

System.out.println();

System.out.println("\nFirst element: "+ fruits.getFirst());

System.out.println("\nLast element: "+ fruits.getLast());

fruits.removeFirst();

fruits.removeLast();

System.out.println("\nElements in linked List: ");

for( String fruit:fruits){

System.out.print(fruit);

}

fruits.set(0, "Pomegranate ");

System.out.println("\n\nElements in linked List: ");

for( String fruit:fruits){

System.out.print(fruit);

}

System.out.println("\n");

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

System.out.print(fruits.get(i));

}

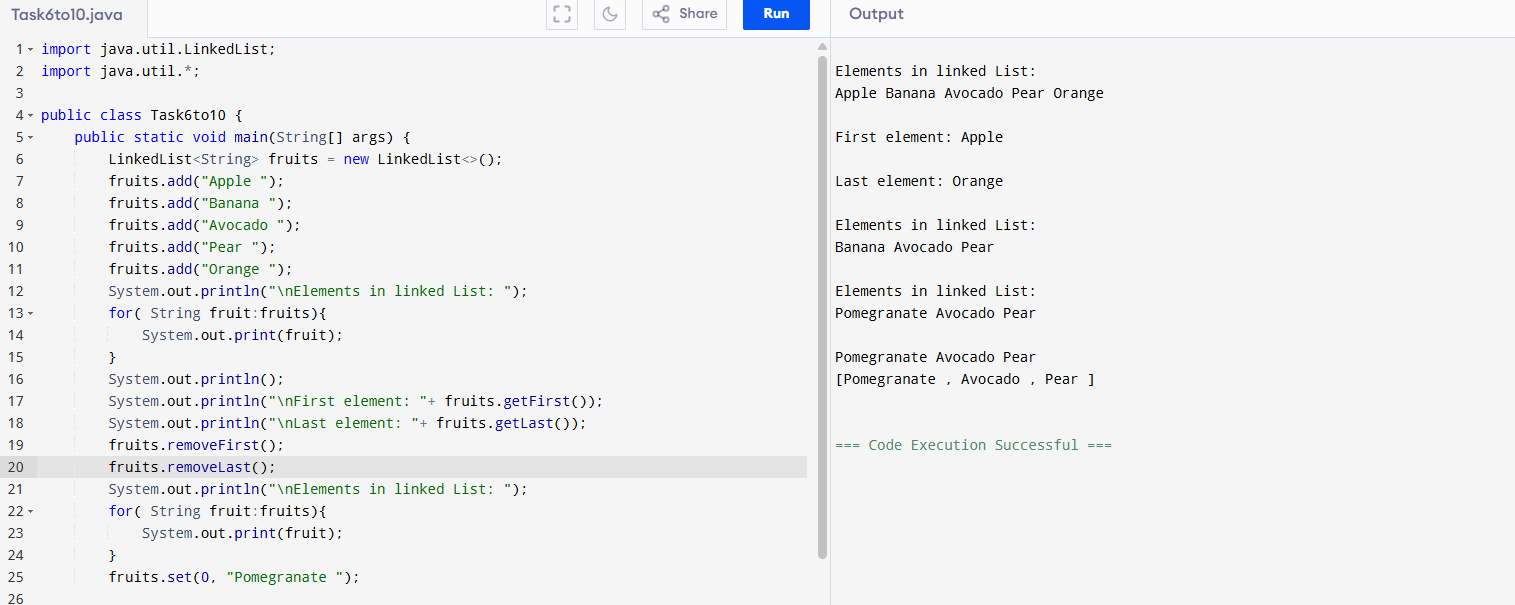
System.out.println();

System.out.println(fruits);

System.out.println();

}

}



Task 11-13:

import java.util.Arrays;

import java.util.LinkedList;

public class Task11to13 {

public static void main(String[] args) {

LinkedList<String> grade8 = new LinkedList<>();

grade8.add("Durga");

grade8.add("Hema");

grade8.add("Mohan");

grade8.add("sudha");

grade8.add("Vivek");

grade8.add("Sandhya");

System.out.println("Original list grade8 printed using Arrays.toString(): ");

Object[] arr = grade8.toArray();System.out.println(Arrays.toString(arr));

LinkedList<String> grade8clone = (LinkedList<String>) grade8.clone();

System.out.println("Clone of List grade8: \n"+grade8clone);

grade8clone.pop();

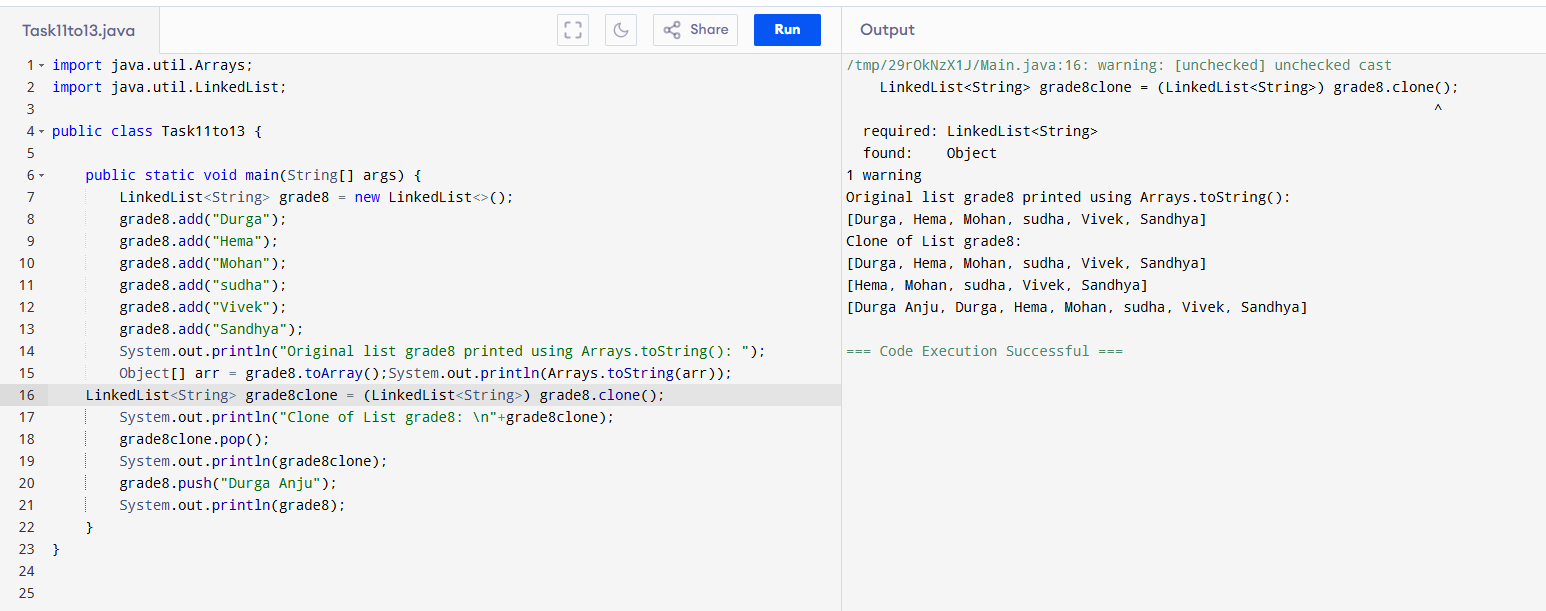
System.out.println(grade8clone);

grade8.push("Durga Anju");

System.out.println(grade8);

}

}



Task14:

import java.util.\*;

public class Task14 {

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();

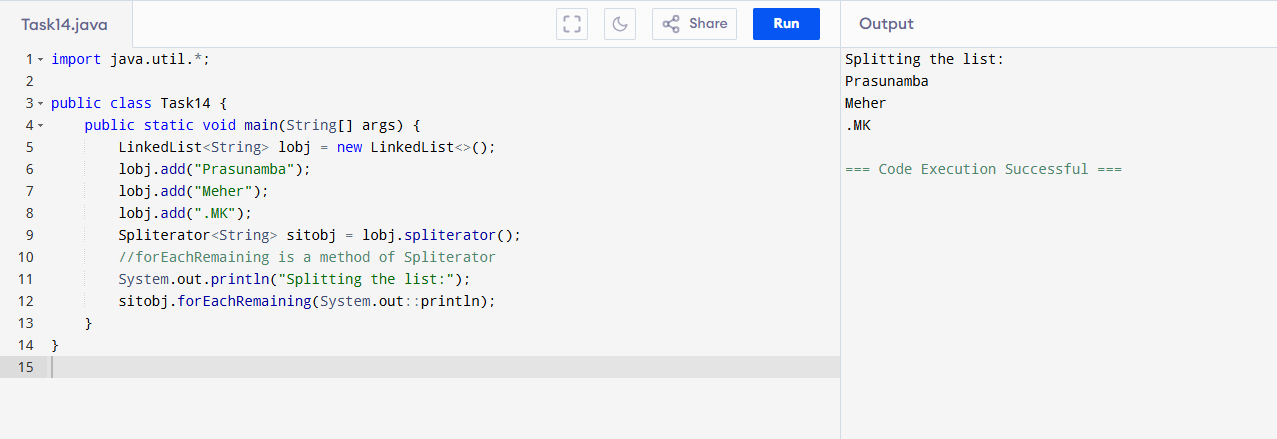
//forEachRemaining is a method of Spliterator

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

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

}

}



Task15:

import java.util.LinkedList;

import java.util.Spliterator;

public class Task15 {

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");

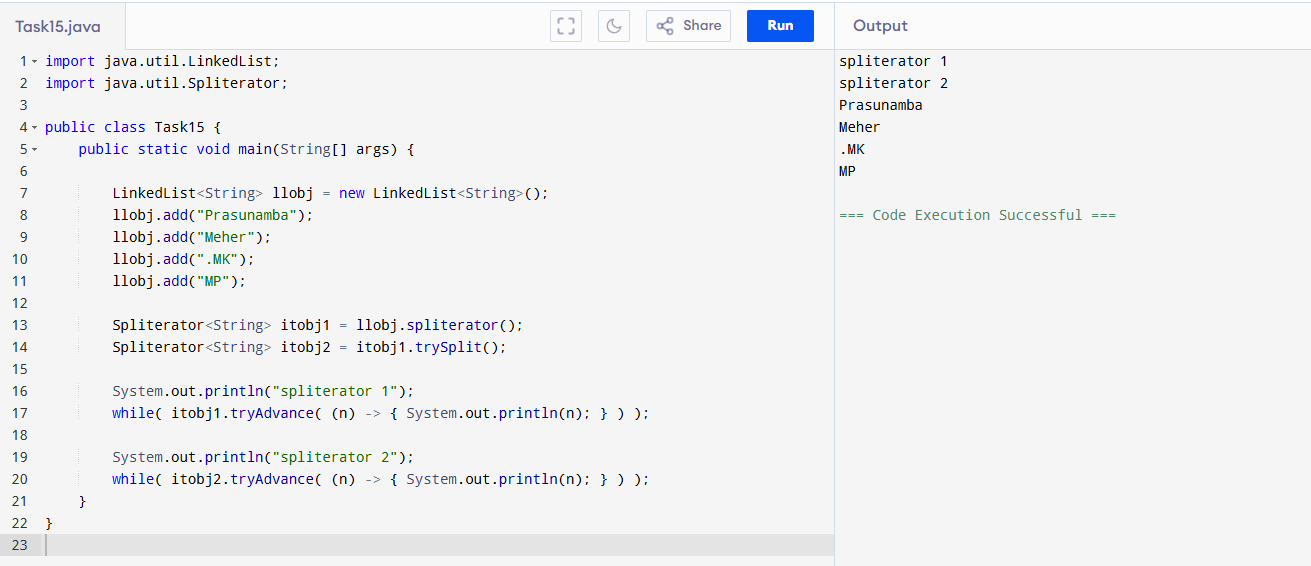
while( itobj1.tryAdvance( (n) -> { System.out.println(n); } ) );

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

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

}

}



Task16:

public class Task16 {

class Node {

int data;

Node next;

Node prev;

public Node(int data) {

this.data = data;

this.next = null;

this.prev = null;

}

}

private Node head;

private Node tail;

public Task16() {

this.head = null;

this.tail = null;

}

public void insertAtEnd(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

newNode.prev = tail;

tail = newNode;

}

System.out.println("Inserted " + value + " at end.");

}

public void deleteByValue(int value) {

if (head == null) {

System.out.println("List is empty. Cannot delete " + value + ".");

return;

}

Node current = head;

if (head.data == value) {

head = head.next;

if (head != null) {

head.prev = null;

} else {

tail = null;

}

System.out.println("Deleted " + value + " from head.");

return;

}

while (current != null && current.data != value) {

current = current.next;

}

if (current == null) {

System.out.println(value + " not found in the list.");

return;

}

if (current.prev != null) {

current.prev.next = current.next;

}

if (current.next != null) {

current.next.prev = current.prev;

}

if (current == tail) {

tail = current.prev;

}

System.out.println("Deleted " + value + ".");

}

public void display() {

Node temp = head;

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

while (temp != null) {

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

temp = temp.next;

}

System.out.println("NULL");

}

public void displayReverse() {

Node temp = tail;

System.out.print("Linked List (Reverse): ");

while (temp != null) {

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

temp = temp.prev;

}

System.out.println("NULL");

}

public static void main(String[] args) {

Task16 list = new Task16();

list.insertAtEnd(10);

list.insertAtEnd(20);

list.insertAtEnd(30);

list.insertAtEnd(40);

list.display();

list.displayReverse();

System.out.println("\n--- Deleting operations ---");

list.deleteByValue(20);

list.display();

list.displayReverse();

list.deleteByValue(10);

list.display();

list.displayReverse();

list.deleteByValue(40);

list.display();

list.displayReverse();

list.deleteByValue(30);

list.display();

list.displayReverse();

System.out.println("\n--- Testing empty list deletion and insertion ---");

list.deleteByValue(50);

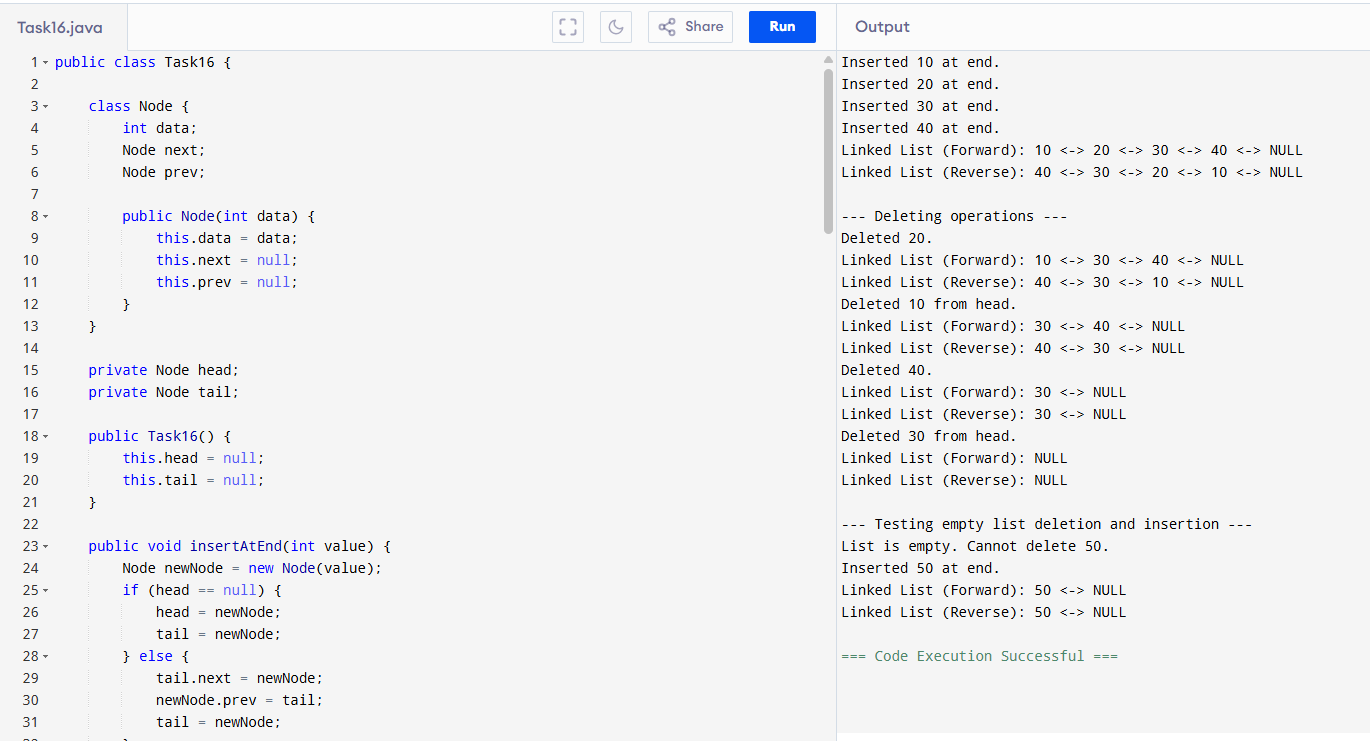
list.insertAtEnd(50);

list.display();

list.displayReverse();

}

}



Task 17-19:

import java.util.HashMap;

public class Task17to19 {

public static void main(String[] args) {

HashMap<String, Integer> hm1 = new HashMap<>(10, 0.75f);

hm1.put("Durga", 1);

System.out.println(hm1);

HashMap<String, Integer>hm2 = new HashMap<String, Integer>(hm1);

System.out.println("Printing copied hashmap: "+hm2);

}

}



Task20:

public class Task21 {

private static class Node {

int data;

Node next;

public Node(int value) {

this.data = value;

this.next = null;

}

}

private Node head;

public Task21() {

this.head = null;

}

public void insertAtEnd(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

newNode.next = head;

} else {

Node temp = head;

while (temp.next != head) {

temp = temp.next;

}

temp.next = newNode;

newNode.next = head;

}

System.out.println("Inserted " + value);

}

public void display() {

if (head == null) {

System.out.println("List is empty.");

return;

}

Node current = head;

System.out.print("Circular Linked List: ");

do {

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

current = current.next;

} while (current != head);

System.out.println(" (Back to Head) " + current.data);

}

public static void main(String[] args) {

Task21 myCircularList = new Task21();

myCircularList.insertAtEnd(10);

myCircularList.display();

myCircularList.insertAtEnd(20);

myCircularList.display();

myCircularList.insertAtEnd(30);

myCircularList.display();

myCircularList.insertAtEnd(40);

myCircularList.display();

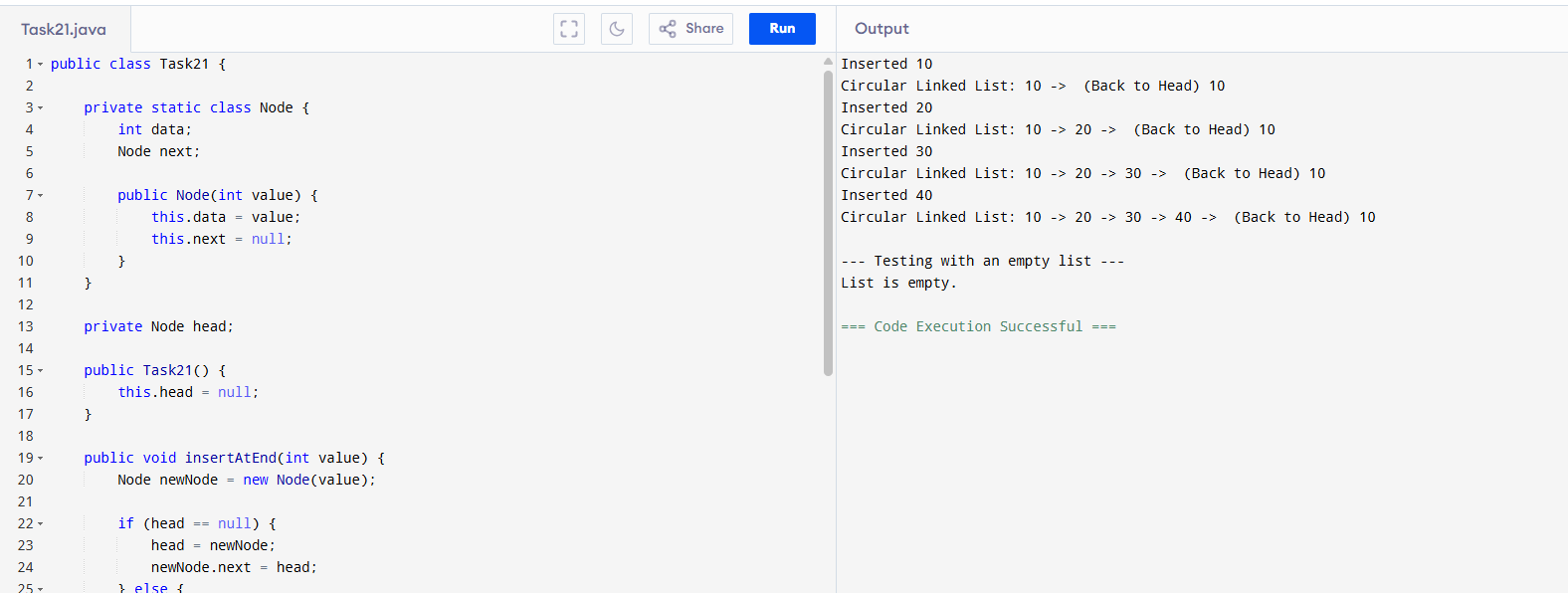
System.out.println("\n--- Testing with an empty list ---");

Task21 emptyList = new Task21();

emptyList.display();

}

}



Home tasks:

Advantages of Linked Lists.

• Linked Lists can have a dynamic size.

• Faster insertions and deletions.

• Efficient memory utilization.

• Can easily be used to implement abstract data types like queues or stacks.

Disadvantages of Linked Lists.

• Each node has reference to other node along with the data, which results in more memory consumption.

• Random access is impossible without traversing the whole List.

• Complexity of implementation. (Reference to other nodes.)

Applications of Linked Lists:

• Implement stacks and queues.

• Used in Dynamic memory allocation

• Image viewers and music players where next and previous buttons are used.

• Web browser’s back and forward functionality.

• Separate chaining in hash tables.

• Adjacency lists in graphs.

• Process scheduling in Operating systems.