JAVA PROGRAMS (5/8/24)

Anugraha N

(192371018)

Q1) CREATE A QUEUE USING LINKED LIST

class Node<T> {

T data;

Node<T> next;

Node(T data) {

this.data = data;

this.next = null;

}

}

public class LinkedListQueue<T> {

private Node<T> front;

private Node<T> rear;

private int size;

public LinkedListQueue() {

front = rear = null;

size = 0;

}

public void enqueue(T data) {

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

if (rear == null) {

front = rear = newNode;

} else {

rear.next = newNode;

rear = newNode;

}

size++;

}

public T dequeue() {

if (front == null) {

throw new IllegalStateException("Queue is empty");

}

T data = front.data;

front = front.next;

if (front == null) {

rear = null;

}

size--;

return data;

}

public T peek() {

if (front == null) {

throw new IllegalStateException("Queue is empty");

}

return front.data;

}

public boolean isEmpty() {

return front == null;

}

public int size() {

return size;

}

public void printQueue() {

Node<T> temp = front;

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

public static void main(String[] args) {

LinkedListQueue<Integer> queue = new LinkedListQueue<>();

queue.enqueue(10);

queue.enqueue(20);

queue.enqueue(30);

queue.enqueue(40);

System.out.println("Queue after enqueuing elements:");

queue.printQueue();

System.out.println("Dequeued element: " + queue.dequeue());

System.out.println("Queue after dequeuing one element:");

queue.printQueue();

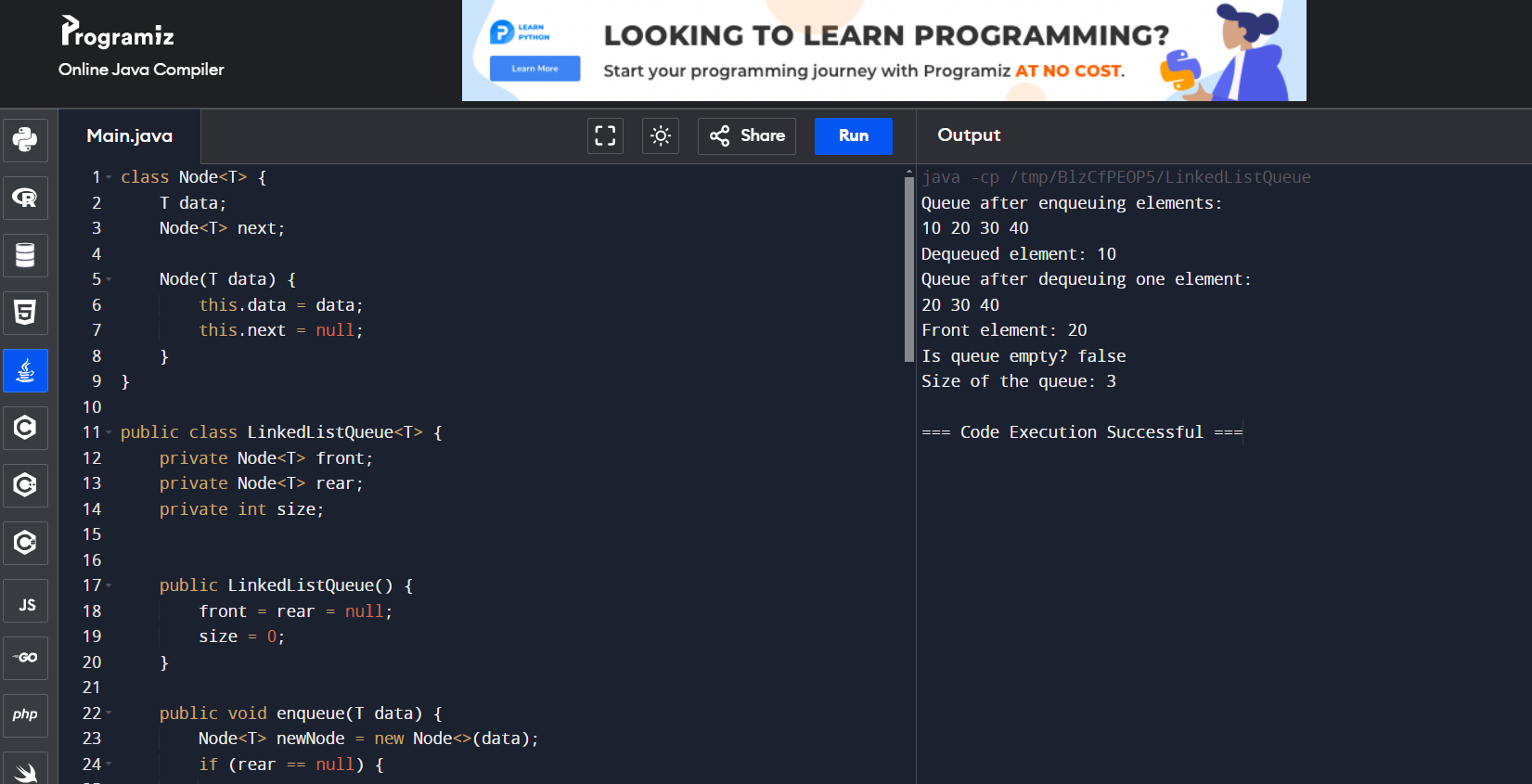
System.out.println("Front element: " + queue.peek());

System.out.println("Is queue empty? " + queue.isEmpty());

System.out.println("Size of the queue: " + queue.size());

}

}



Q2) CREATE STACK USING LINKED LIST

class Node<T> {

T data;

Node<T> next;

Node(T data) {

this.data = data;

this.next = null;

}

}

public class LinkedListStack<T> {

private Node<T> top;

private int size;

public LinkedListStack() {

top = null;

size = 0;

}

public void push(T data) {

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

newNode.next = top;

top = newNode;

size++;

}

public T pop() {

if (top == null) {

throw new IllegalStateException("Stack is empty");

}

T data = top.data;

top = top.next;

size--;

return data;

}

public T peek() {

if (top == null) {

throw new IllegalStateException("Stack is empty");

}

return top.data;

}

public boolean isEmpty() {

return top == null;

}

public int size() {

return size;

}

public void printStack() {

Node<T> temp = top;

while (temp != null) {

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

temp = temp.next;

}

System.out.println();

}

public static void main(String[] args) {

LinkedListStack<Integer> stack = new LinkedListStack<>();

stack.push(10);

stack.push(20);

stack.push(30);

stack.push(40);

System.out.println("Stack after pushing elements:");

stack.printStack();

System.out.println("Popped element: " + stack.pop());

System.out.println("Stack after popping one element:");

stack.printStack();

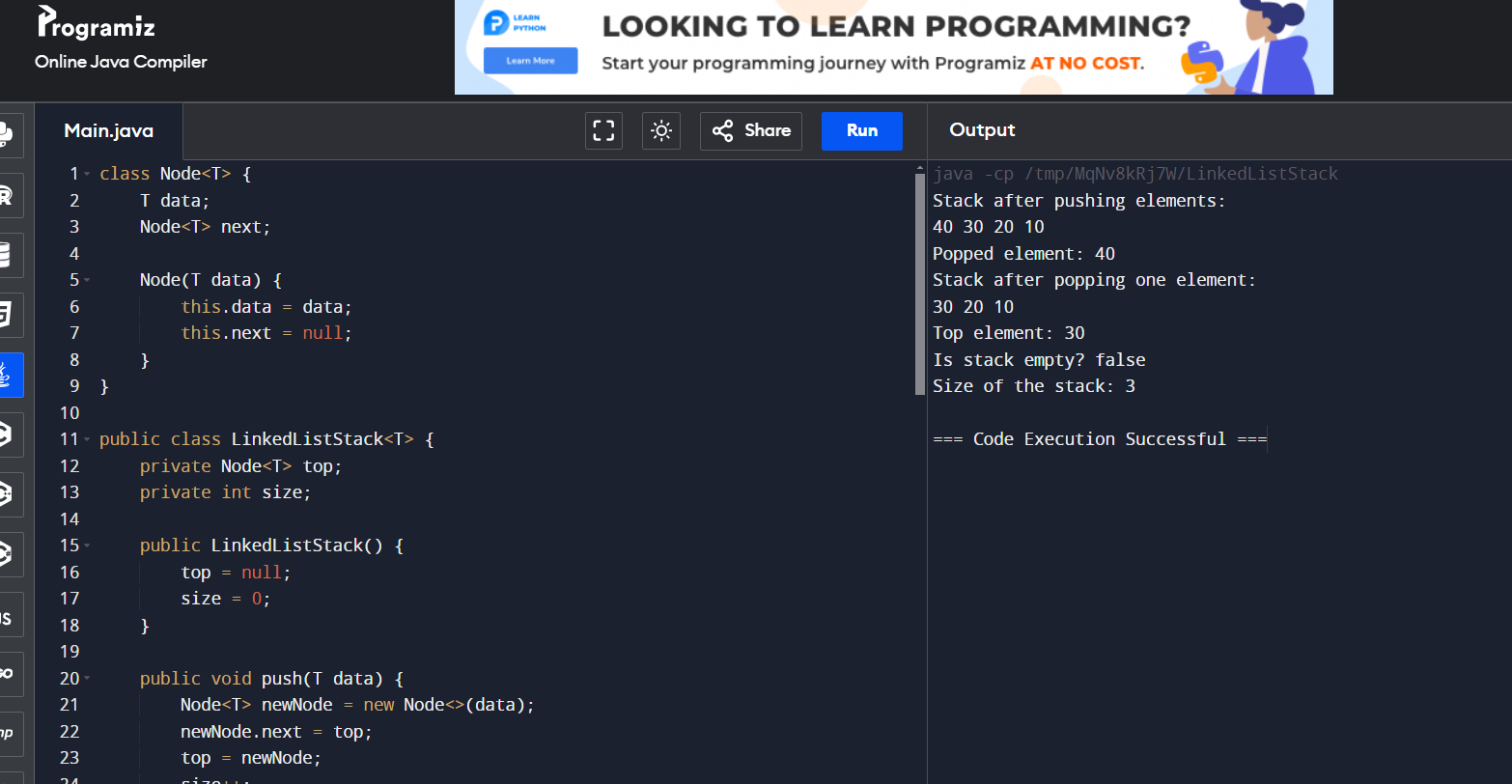
System.out.println("Top element: " + stack.peek());

System.out.println("Is stack empty? " + stack.isEmpty());

System.out.println("Size of the stack: " + stack.size());

}

}



Q3)

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class Main {

public static void main(String[] args) {

List<Mobile> mobiles = new ArrayList<>();

mobiles.add(new Mobile("Model A", 299.99));

mobiles.add(new Mobile("Model B", 499.99));

mobiles.add(new Mobile("Model C", 199.99));

mobiles.add(new Mobile("Model D", 349.99));

System.out.println("Before sorting:");

for (Mobile mobile : mobiles) {

System.out.println(mobile);

}

Collections.sort(mobiles);

System.out.println("\nAfter sorting by cost:");

for (Mobile mobile : mobiles) {

System.out.println(mobile);

}

}

static class Mobile implements Comparable<Mobile> {

private String name;

private double cost;

public Mobile(String name, double cost) {

this.name = name;

this.cost = cost;

}

public String getName() {

return name;

}

public double getCost() {

return cost;

}

@Override

public int compareTo(Mobile other) {

return Double.compare(this.cost, other.cost);

}

@Override

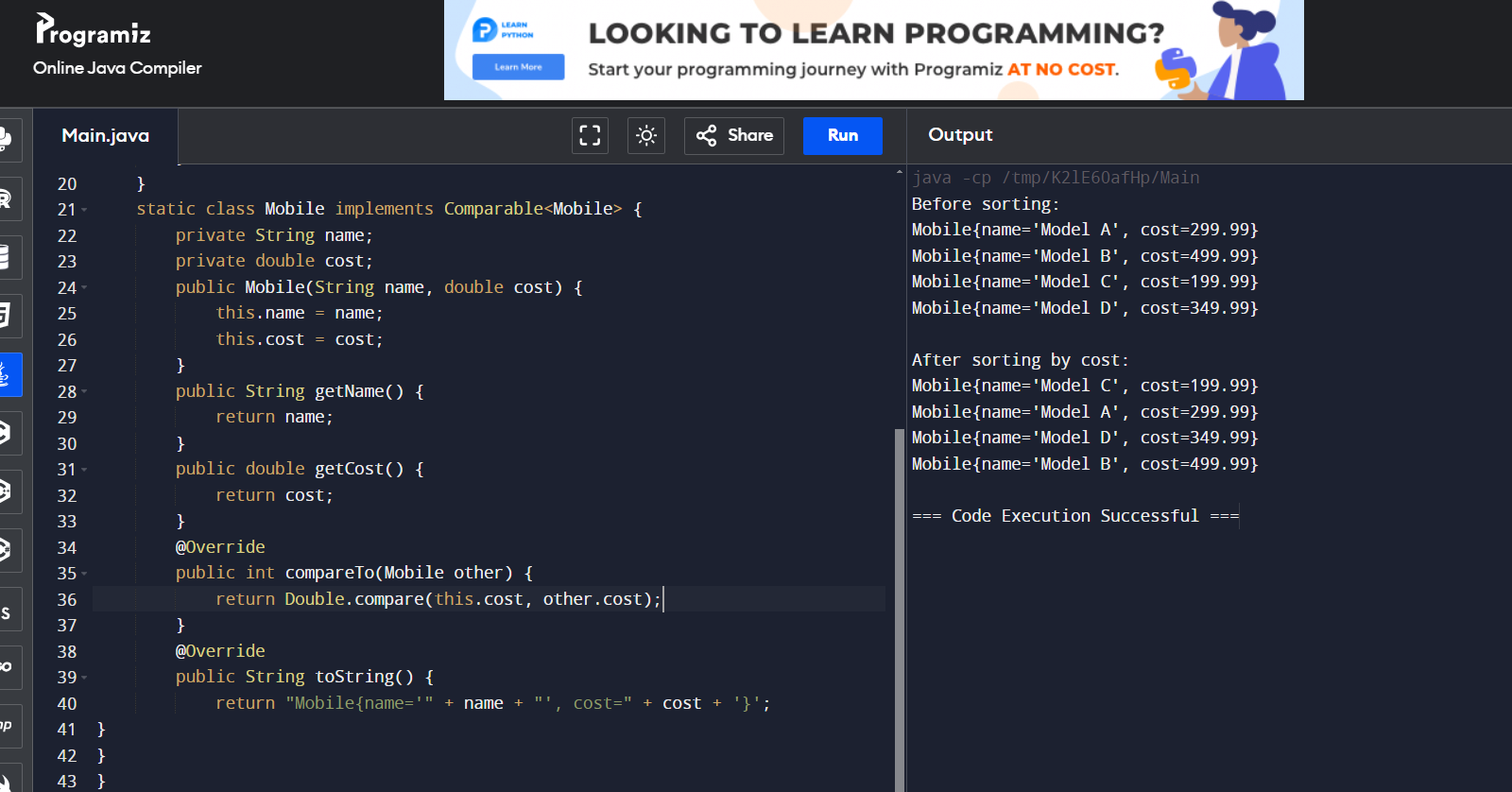
public String toString() {

return "Mobile{name='" + name + "', cost=" + cost + '}';

}

}

}



Q4)

import java.util.HashMap;

import java.util.Map;

public class HashMapExample {

public static void main(String[] args) {

Map<String, Integer> ageMap = new HashMap<>();

ageMap.put("Alice", 30);

ageMap.put("Bob", 25);

ageMap.put("Charlie", 35);

ageMap.put("Diana", 40);

System.out.println("Alice's age: " + ageMap.get("Alice"));

System.out.println("Bob's age: " + ageMap.get("Bob"));

if (ageMap.containsKey("Charlie")) {

System.out.println("Charlie's age: " + ageMap.get("Charlie"));

}

ageMap.remove("Diana");

System.out.println("\nCurrent map entries:");

for (Map.Entry<String, Integer> entry : ageMap.entrySet()) {

System.out.println(entry.getKey() + ": " + entry.getValue());

}

System.out.println("\nKeys:");

for (String key : ageMap.keySet()) {

System.out.println(key);

}

System.out.println("\nValues:");

for (Integer value : ageMap.values()) {

System.out.println(value);

}

System.out.println("\nSize of the map: " + ageMap.size());

}

}

