**3(a)**

import java.util.Queue;

import java.util.LinkedList;

public class StackUsingTwoQueues {

Queue<Integer> q1 = new LinkedList<>();

Queue<Integer> q2 = new LinkedList<>();

void push(int x) {

q2.add(x);

while (!q1.isEmpty()) q2.add(q1.poll());

Queue<Integer> temp = q1; q1 = q2; q2 = temp;

}

int pop() {

if (q1.isEmpty()) return -1;

return q1.poll();

}

int top() {

if (q1.isEmpty()) return -1;

return q1.peek();

}

boolean isEmpty() {

return q1.isEmpty();

}

public static void main(String[] args) {

StackUsingTwoQueues stack = new StackUsingTwoQueues();

stack.push(10);

stack.push(20);

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

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

}

}

**3(b)**

import java.util.\*;

public class BagOfNumbers {

public static void main(String[] args) {

ArrayList<Integer> bag = new ArrayList<>();

System.out.println("Adding numbers: 5, 10, 5, 20");

Collections.addAll(bag, 5, 10, 5, 20);

System.out.println("Bag contents: " + bag);

System.out.println("Count occurrences of 5: " + Collections.frequency(bag, 5));

System.out.println("Removing 5 from the bag...");

bag.remove(Integer.valueOf(5));

System.out.println("Bag contents: " + bag);

System.out.println("Bag size: " + bag.size());

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

System.out.println("Removing all numbers...");

bag.clear();

System.out.println("Is the bag empty now? " + bag.isEmpty());

}

}

**3©**

import java.util.\*;

public class DiskTower {

static void solve(int day, int n, int[] disks, PriorityQueue<Integer> pq, int[] max) {

if (day == n) return;

pq.add(disks[day]);

System.out.print("Day " + (day + 1) + ": ");

while (!pq.isEmpty() && pq.peek() == max[0]) {

System.out.print(pq.poll() + " ");

max[0]--;

}

System.out.println();

solve(day + 1, n, disks, pq, max);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of disks (days): ");

int n = sc.nextInt();

int[] disks = new int[n];

System.out.println("Enter the disk sizes:");

for (int i = 0; i < n; i++) disks[i] = sc.nextInt();

PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

solve(0, n, disks, pq, new int[]{n});

    }

}