**ASSIGNMENT -1**

**Object Oriented Programming**

Pratham Kandari

500097663

Batch: 7

1. John has discovered various rocks. Each rock is composed of various elements, and each element is represented by a lower-case Latin letter from ‘a’ to 'z’. An element can be present multiple times in a rock. An element is called a gem-element if it occurs at least once in each of the rocks. Given the list of N rocks with their compositions, display the number of gem-elements that exist in those rocks.

Ans:

import java.util.\*;

class John

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter the total number of rocks present");

int N=sc.nextInt();

String str;

int a[]=new int[26];

int b[];

int i=0;

for(i=0;i<N;i++)

{

b=new int[26];

str=sc.next();

for(char c: str.toCharArray())

{

b[c-97]+=1;

if(b[c-97]==1)

a[c-97]+=1;

}

b=null;

}

int sum=0;

for(i=0;i<26;i++)

{

if(a[i]==N)

sum=sum+1;

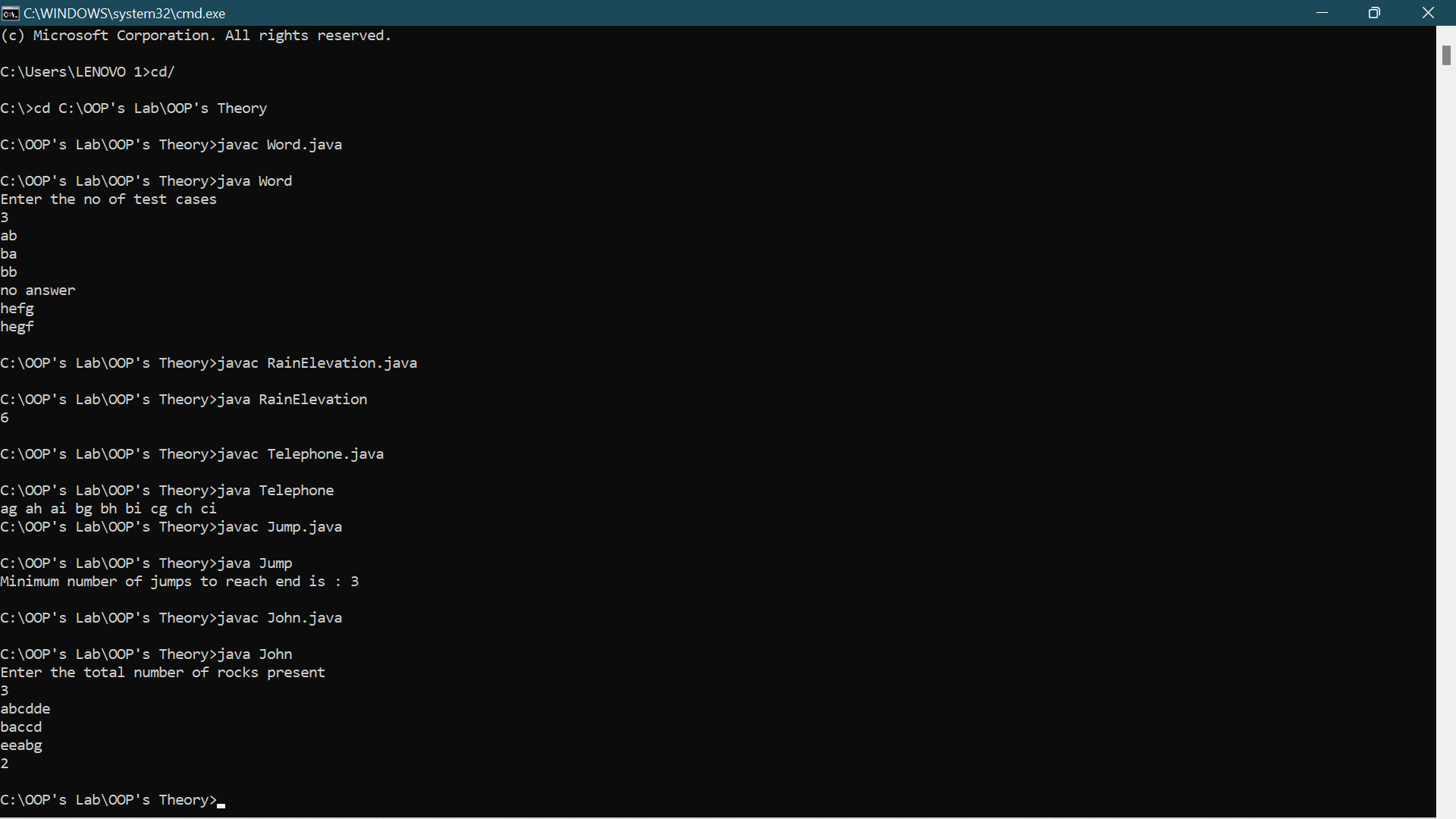
}

a=null; //Garbage Collector

System.out.println(sum);

}

}­



1. Given a word w, rearrange the letters of w to construct another word s in such a way that s is lexicographically greater than w. In case of multiple possible answers, find the lexicographically one.

Ans:

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

class Word

{

public static void check(char str[])

{

int i=0,flag=0;

loop:

for(i=str.length-1;i>0;i--)

{

if(str[i]>str[i-1])

{

int j = str.length-1;

while(flag==0 && j!=i-1)

{

if(str[i-1]<str[j])

{

char t = str[j];

str[j]=str[i-1];

str[i-1]=t;

flag=1;

break loop;

}

j--;

}

}

}

Arrays.sort(str,i,str.length);

if(flag==0)

{

System.out.println("no answer");

}

else

{

System.out.println(str);

}

}

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter the no of test cases");

int T=sc.nextInt();

char word[];

int i=0;

for(i=0;i<T;i++)

{

word=sc.next().toCharArray();

check(word);

}

}

}

Graphical user interface

Description automatically generated with medium confidence

1. Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Ans:

class RainElevation

{

static int getWater(int a[], int n)

{

int res = 0;

for(int i = 1; i < n - 1; i++)

{

int lMax = a[i];

for(int j = 0; j < i; j++)

lMax = Math.max(lMax, a[j]);

int rMax = a[i];

for(int j = i + 1; j < n; j++)

rMax = Math.max(rMax, a[j]);

res = res + (Math.min(lMax, rMax) - a[i]);

}

return res;

}

public static void main(String args[])

{

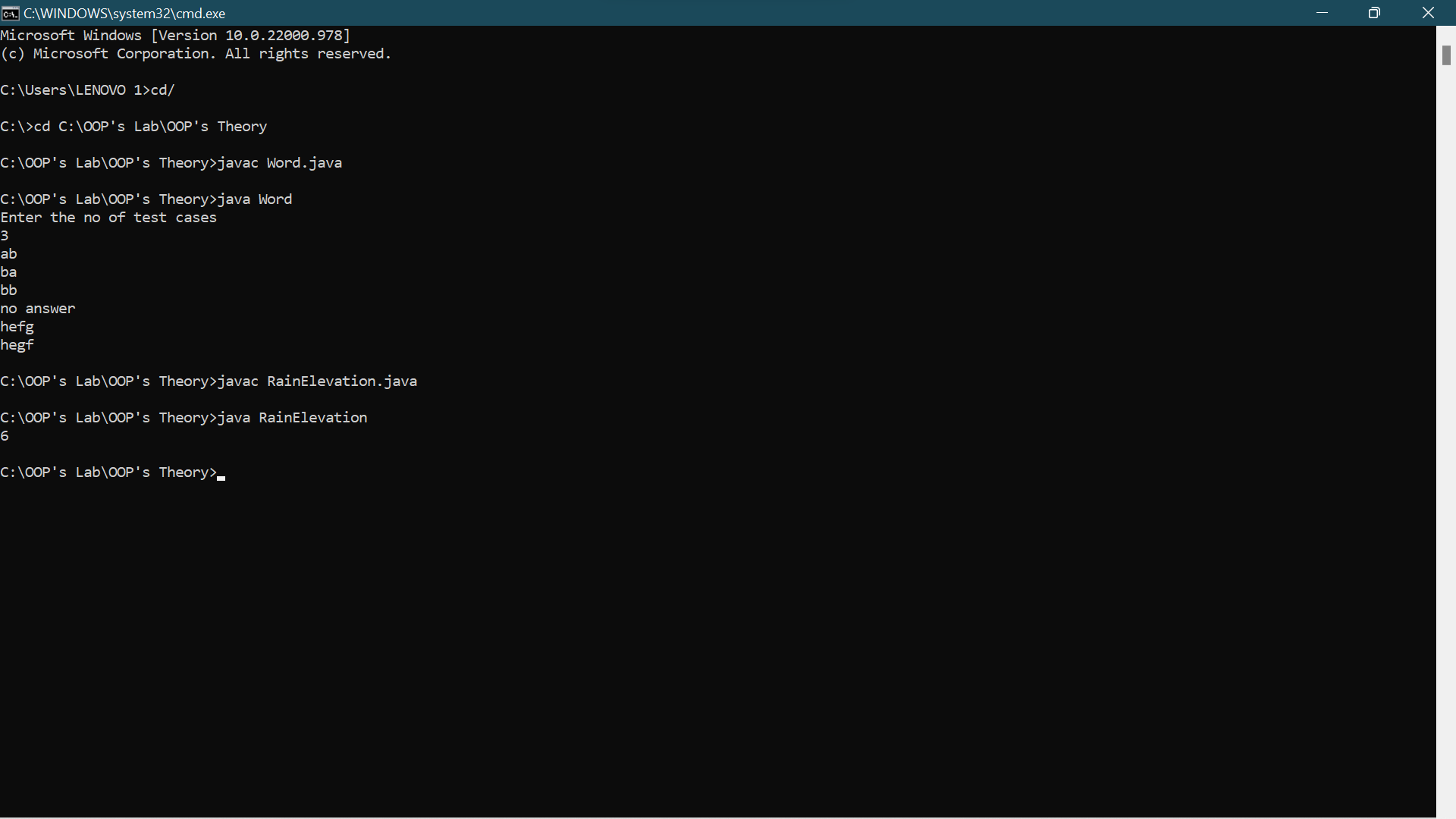
int height[] = {0,1,0,2,1,0,1,3,2,1,2,1};

int n = 12;

System.out.println(getWater(height, n));

}

}



1. Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order. A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Ans:

import java.util.\*;

class Telephone{

static ArrayList<String>

letterCombinationsUtil(int number[], int n, String table[])

{

ArrayList<String> list = new ArrayList<>();

Queue<String> q = new LinkedList<>();

q.add("");

while (!q.isEmpty()) {

String s = q.remove();

if (s.length() == n)

list.add(s);

else {

String val = table[number[s.length()]];

for (int i = 0; i < val.length(); i++)

{

q.add(s + val.charAt(i));

}

}

}

return list;

}

static void letterCombinations(int number[], int n)

{

String table[] = { "0", "1", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz" };

ArrayList<String> list = letterCombinationsUtil(number, n, table);

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

{

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

}

}

public static void main(String args[])

{

int number[] = {2,4};

int n = number.length;

letterCombinations(number, n);

}

}

Text

Description automatically generated

1. Given an array of non-negative integers nums, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Your goal is to reach the last index in the minimum number of jumps. You can assume that you can always reach the last index.

Ans:

class Jump

{

static int jumps(int arr[], int n)

{

int jumps[] = new int[n];

int i, j;

if (n == 0 || arr[0] == 0)

return Integer.MAX\_VALUE;

jumps[0] = 0;

for (i = 1; i < n; i++)

{

jumps[i] = Integer.MAX\_VALUE;

for (j = 0; j < i; j++)

{

if (i <= j + arr[j] && jumps[j] != Integer.MAX\_VALUE)

{

jumps[i] = Math.min(jumps[i], jumps[j] + 1);

break;

}

}

}

return jumps[n - 1];

}

public static void main(String[] args)

{

int arr[] = { 1, 3, 6, 1, 0, 9 };

int n=arr.length;

System.out.println("Minimum number of jumps to reach end is : "+ jumps(arr, n));

}

}