**1. Check Sum of Odd Digits**

Write a program to read a number, calculate the sum of odd digits (values) present in the given number.

Include a class UserMainCode with a static method checkSum which accepts a positive integer . The return type should be 1 if the sum is odd. In case the sum is even return -1 as output.

Create a class Main which would get the input as a positive integer and call the static method checkSum present in the UserMainCode.

Input and Output Format:

Input consists of a positive integer n.

Refer sample output for formatting specifications.

**Sample Input 1:**

56895

**Sample Output 1:**

Sum of odd digits is odd.

**Sample Input 2:**

84228

**Sample Output 2:**

Sum of odd digits is even.

public class UserMainCode

{

            public static int checkSum(int num)

            {

                        int n1,n2=0,n3 = 0;

                        while(num!=0)

                        {

                                    n1=num%10;

                                    if(n1%2!=0)

                                                n2 = n2+n1;

                                                num = num/10;

                                    if(num%2==0)

                                                return -1;

                                    else

                                                return 1;

                        }

                        return n3;

            }

}

**2. Number Validation**

Write a program to read a string of 10 digit number , check whether the string contains a 10 digit number in the format XXX-XXX-XXXX where 'X' is a digit.

Include a class UserMainCode with a static method validateNumber which accepts a string as input .

The return type of the output should be 1 if the string meets the above specified format . In case the number does not meet the specified format then return -1 as output.

Create a class Main which would get the input as a String of numbers and call the static method validateNumber present in the UserMainCode.

Input and Output Format:

Input consists of a string.

Output is a string specifying the given string is valid or not .

Refer sample output for formatting specifications.

**Sample Input 1:**

123-456-7895

**Sample Output 1:**

Valid number format

**Sample Input 2:**

-123-12344322

**Sample Output 2:**

Invalid number format

public class UserMainCode {

            public static int numberValidation(String s1) {

                        if (s1.matches("[0-9]{3}[-]{1}[0-9]{3}[-]{1}[0-9]{4}")) {

                                    return 1;

                        } else {

                                    return -1;

                        }

            }

}

**3. Sum of Squares of Even Digits**

Write a program to read a number , calculate the sum of squares of even digits (values) present in the given number.

Include a class UserMainCode with a static method sumOfSquaresOfEvenDigits which accepts a positive integer . The return type (integer) should be the sum of squares of the even digits.

Create a class Main which would get the input as a positive integer and call the static method sumOfSquaresOfEvenDigits present in the UserMainCode.

Input and Output Format:

Input consists of a positive integer n.

Output is a single integer .

Refer sample output for formatting specifications.

**Sample Input 1:**

56895

**Sample Output 1:**

100

public class UserMainCode {

            public static int sumSqOfEven(int a) {

                        int rem, sum = 0, square;

                        while (a != 0) {

                                    rem = a % 10;

                                    if (rem % 2 == 0) {

                                                square = rem \* rem;

                                                sum = sum + square;

                                    }

                                    a = a / 10;

                        }

                        return sum;

            }

}

**4. Fetching Middle Characters from String**

Write a program to read a string of even length and to fetch two middle most characters from the input string and return it as string output.

Include a class UserMainCode with a static method getMiddleChars which accepts a string of even length as input . The return type is a string which should be the middle characters of the string.

Create a class Main which would get the input as a string and call the static method getMiddleChars present in the UserMainCode.

Input and Output Format:

Input consists of a string of even length.

Output is a string .

Refer sample output for formatting specifications.

**Sample Input 1:**

this

**Sample Output 1:**

hi

**Sample Input 1:**

Hell

**Sample Output 1:**

el

public class UserMainCode {

            public static String fetchMiddle(String s1) {

                        String res = " ";

                        if (s1.length() % 2 == 0) {

                                    if (s1.length() > 2) {

                                                res = s1.substring((s1.length() / 2) - 1, (s1.length() / 2) + 1);

                                    }

                        }

                        return res;

            }

}

**5. Check Characters in a String**

Write a program to read a string  and to test whether first and last character are same. The string is said to be be valid if the 1st and last character are the same. Else the string is said to be invalid.

Include a class UserMainCode with a static method checkCharacters which accepts a string as input .

The return type of this method is an int.  Output should be 1 if the first character and last character are same . If they are different then return -1 as output.

Create a class Main which would get the input as a string and call the static method checkCharacters present in the UserMainCode.

Input and Output Format:

Input consists of a string.

Output is a string saying characters are same or not .

Refer sample output for formatting specifications.

**Sample Input 1:**

the picture was great

**Sample Output 1:**

Valid

**Sample Input 1:**

this

**Sample Output 1:**

Invalid

public class UserMainCode {

            public static int checkChar(String s) {

                        StringBuffer sb = new StringBuffer(s);

                        sb.reverse();

                        String s1 = sb.toString();

                        if (s.charAt(0) == s1.charAt(0)) {

                                    return 1;

                        } else {

                                    return -1;

                        }

            }

}

**6. Forming New Word from a String**

Write a program to read a string and a positive integer n as input and construct a string with first n and last n characters in the given string.

Include a class UserMainCode with a static method formNewWord which accepts a string and positive integer .

The return type of the output should be a string (value) of first n character and last n character.

Create a class Main which would get the input as a string and integer n and call the static method formNewWord present in the UserMainCode.

Input and Output Format:

Input consists of a string of even length.

Output is a string .

Note: The given string length must be >=2n.

Refer sample output for formatting specifications.

Sample Input 1:

California

3

**Sample Output 1:**

Calnia

**Sample Input 2:**

this

1

**Sample Output 2:**

ts

**public** **class** UserMainCode {

**public** **static** String formNewWord(String s1, **int** a) {

                        StringBuffer sb = **new** StringBuffer();

**int** b = s1.length();

**int** c = b - a;

**if** (b > 2 \* a) {

                                    sb.append(s1.substring(0, a)).append(s1.substring(c));

                        }

**return** sb.toString();

            }

}

**7. Reversing a Number**

Write a program to read a positive number as input and to get the reverse of the given number and return it as output.

Include a class UserMainCode with a static method reverseNumber which accepts a positive integer .

The return type is an integer value which is the reverse of the given number.

Create a Main class which gets the input as a integer and call the static method reverseNumber present in the UserMainCode

Input and Output Format:

Input consists of a positive integer.

Output is an integer .

Refer sample output for formatting specifications.

**Sample Input 1:**

543

**Sample Output 1:**

345

**Sample Input 1:**

1111

**Sample Output 1:**

1111

**public** **class** UserMainCode {

**public** **static** **int** reverse(**int** a) {

**int** rem, rev = 0;

**while** (a != 0) {

                                    rem = a % 10;

                                    rev = rev \* 10 + rem;

                                    a /= 10;

                        }

**return** rev;

            }

**8. Array List Sorting and Merging**

Write a code to read two int array lists of size 5 each as input and to merge the two arrayLists, sort the merged arraylist in ascending order and fetch the elements at 2nd, 6th and 8th index into a new arrayList and return the final ArrayList.

Include a class UserMainCode with a static method sortMergedArrayList which accepts 2 ArrayLists.

The return type is an ArrayList with elements from 2,6 and 8th index position .Array index starts from position 0.

Create a Main class which gets two array list of size 5 as input and call the static method sortMergedArrayList present in the UserMainCode.

Input and Output Format:

Input consists of two array lists of size 5.

Output is an array list .

Note - The first element is at index 0.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

1

17

11

19

5

2

7

6

20

**Sample Output 1:**

3

11

19

**Sample Input 2:**

1

2

3

4

5

6

7

8

9

10

**Sample Output 2:**

3

7

9

**public** **class** Main {

**public** **static** **void** main(String[] args) {

                        Scanner sc = **new** Scanner(System.*in*);

**int** a = 5;

                        ArrayList<Integer> al1 = **new** ArrayList<Integer>();

                        ArrayList<Integer> al2 = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < a; i++) {

                                    al1.add(sc.nextInt());

                        }

**for** (**int** i = 0; i < a; i++) {

                                    al2.add(sc.nextInt());

                        }

                        al1.addAll(al2);

                        ArrayList<Integer> s = UserMainCode.*sortMergedArrayList*(al1, al2);

                        Integer t[] = **new** Integer[s.size()];

                        s.toArray(t);

**for** (**int** i = 0; i < t.length; i++) {

                                    System.*out*.println(t[i]);

                        }

            }

}

**public** **class** UserMainCode {

**public** **static** ArrayList<Integer> sortMergedArrayList(

                                    ArrayList<Integer> al1, ArrayList<Integer> al2) {

                        ArrayList<Integer> al3 = **new** ArrayList<Integer>();

                        Collections.*sort*(al1);

**for** (**int** i = 0; i < al1.size(); i++) {

**if** (i == 2 || i == 6 || i == 8) {

                                                al3.add(al1.get(i));

                                    }

                        }

**return** al3;

            }

}

**9. Validating Date Format**

Obtain a date string in the format dd/mm/yyyy. Write code to validate the given date against the given format.

Include a class UserMainCode with a static method validateDate which accepts a string .

The return type of the validateDate method is 1 if the given date format matches the specified format , If the validation fails return the output as -1.

Create a Main class which gets date string as an input and call the static method validateDate present in the UserMainCode.

Input and Output Format:

Input is a string .

Refer sample output for formatting specifications

Sample Input 1:

12/06/1987

**Sample Output 1:**

Valid date format

Sample Input 2:

03/1/1987

**Sample Output 2:**

Invalid date format

**public** **class** UserMainCode {

**public** **static** **int** dateValidate(String s1) {

**int** res = 0;

**if** (s1.matches("[0-9]{2}/[0-9]{2}/[0-9]{4}")) {

                                    SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/yyyy");

                                    sdf.setLenient(**false**);

**try** {

                                                Date d = sdf.parse(s1);

                                                res = 1;

                                    } **catch** (ParseException e) {

                                                res = -1;

                                    }

                        }

**return** res;

            }

}

**10. Validate Time**

Obtain a time string as input in the following format 'hh:mm am' or 'hh:mm pm'. Write code to validate it using the following rules:

- It should be a valid time in 12 hrs format

- It should have case insensitive AM or PM

Include a class UserMainCode with a static method validateTime which accepts a string.

If the given time is as per the given rules then return 1 else return -1.If the value returned is 1 then print as valid time else print as Invalid time.

Create a Main class which gets time(string value) as an input and call the static method validateTime present in the UserMainCode.

**Input and Output Format:**

Input is a string .

Output is a string .

**Sample Input 1:**

09:59 pm

**Sample Output 1:**

Valid time

**Sample Input 2:**

10:70 AM

**Sample Output 2:**

Invalid time

**public** **class** UserMainCode {

**public** **static** **int** validateTime(String s1) {

**int** out = 0;

                        String s[] = s1.split(":");

**for** (**int** i = 0; i < s.length; i++) {

**if** (s.length == 2) {

                                                SimpleDateFormat sdf = **new** SimpleDateFormat("hh:mm a");

                                                sdf.setLenient(**false**);

**try** {

                                                            Date d = sdf.parse(s1);

                                                            out = 1;

                                                } **catch** (Exception e) {

                                                            out = -1;

                                                }

                                    }

                        }

**return** out;56

            }

}

**11. Start Case**

Write a program to read a sentence in string variable and convert the first letter of each word to capital case. Print the final string.

Note: - Only the first letter in each word should be in capital case in final string.

Include a class UserMainCode with a static method printCapitalized which accepts a string. The return type (String) should return the capitalized string.

Create a Class Main which would be used to accept a string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a strings.

Output consists of a String (capitalized string).

Refer sample output for formatting specifications.

**Sample Input:**

Now is the time to act!

**Sample Output:**

Now Is The Time To Act!

**public** **class** UserMainCode {

**public** **static** String startCase(String s1) {

                        String[] arr = s1.split(" ");

                        StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < arr.length; i++) {

                                    sb.append(Character.*toUpperCase*(arr[i].charAt(0)))

                                                            .append(arr[i].substring(1)).append(" ");

                        }

**return** sb.toString().trim();

            }

}

**12. Maximum Difference**

Write a program to read an integer array and find the index of larger number of the two adjacent numbers with largest difference. Print the index.

Include a class UserMainCode with a static method findMaxDistance which accepts an integer array and the number of elements in the array. The return type (Integer) should return index.

Create a Class Main which would be used to accept an integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of n+1 integers, where n corresponds the size of the array followed by n integers.

Output consists of an Integer (index).

Refer sample output for formatting specifications.

**Sample Input :**

6

4

8

6

1

9

4

**Sample Output :**

4

[In the sequence 4 8 6 1 9 4 the maximum distance is 8 (between 1 and 9). The function should return the index of the greatest of two. In this case it is 9 (which is at index 4). output = 4.]

**public** **class** UserMainCode {

**public** **static** **int** maxDiff(**int**[] a) {

**int** max = 0, n1 = 0;

**int**[] diff = **new** **int**[a.length - 1];

**for** (**int** i = 0; i < a.length - 1; i++) {

                                    diff[i] = (**int**) (a[i] - a[i + 1]);

**if** (diff[i] > 0) {

**if** (diff[i] > max) {

                                                            max = diff[i];

                                                            n1 = i;

                                                }

                                    } **else** {

**if** (Math.*abs*(diff[i]) > max) {

                                                            max = diff[i];

                                                            n1 = i + 1;

                                                }

                                    }

                        }

**return** n1;

            }

}

**13. Unique Even Sum**

Write a program to read an array, eliminiate duplicate elements and calculate the sum of even numbers (values) present in the array.

Include a class UserMainCode with a static method addUniqueEven which accepts a single integer array. The return type (integer) should be the sum of the even numbers. In case there is no even number it should return -1.

Create a Class Main which would be used to accept Input array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of n+1 integers. The first integer corresponds to n, the number of elements in the array. The next 'n' integers correspond to the elements in the array.

In case there is no even integer in the input array, print no even numbers as output. Else print the sum.

Refer sample output for formatting specifications.

Assume that the maximum number of elements in the array is 20.

**Sample Input 1:**

4

2

5

1

4

**Sample Output 1:**

6

**Sample Input 2:**

3

1

1

1

**Sample Output 2:**

no even numbers

**public** **class** UserMainCode {

**public** **static** **int** uniqueEvenSum(**int** a[]) {

**int** sum = 0, count = 0;

                        ArrayList<Integer> al = **new** ArrayList<Integer>();

                        al.add(a[0]);

**for** (**int** i = 1; i < a.length; i++) {

**if** (!al.contains(a[i])) {

                                                al.add(a[i]);

                                    }

                        }

**for** (**int** j = 0; j < al.size(); j++) {

**if** (al.get(j) % 2 == 0) {

                                                sum = sum + al.get(j);

                                    } **else** {

                                                count++;

                                    }

                        }

**if** (count == al.size())

**return** -1;

**else**

**return** sum;

            }

}

**14. Palindrome - In Range**

Write a program to input two integers, which corresponds to the lower limit and upper limit respectively, and find the sum of all palindrome numbers present in the range including the two numbers. Print the sum.

Include a class UserMainCode with a static method addPalindromes which accepts two integers. The return type (Integer) should return the sum if the palindromes are present, else return 0.

Create a Class Main which would be used to accept two integer and call the static method present in UserMainCode.

Note1 : A palindrome number is a number which remains same after reversing its digits.

Note2 : A single digit number is not considered as palindrome.

Input and Output Format:

Input consists of 2 integers, which corresponds to the lower limit and upper limit respectively.

Output consists of an Integer (sum of palindromes).

Refer sample output for formatting specifications.

**Sample Input :**

130

150

**Sample Output :**

272

(131+141 = 272)

**public** **class** UserMainCode {

**public** **static** **int** palindromeInrange(**int** ll, **int** ul) {

**int** rem, sum = 0, rev = 0, temp = 0;

**for** (**int** i = ll; i < ul; i++) {

                                    rev = 0;

                                    temp = i;

**while** (temp != 0) {

                                                rem = temp % 10;

                                                rev = rev \* 10 + rem;

                                                temp = temp / 10;

                                    }

**if** (rev == i) {

                                                sum = sum + rev;

                                    }

                        }

**return** sum;

            }

}

**15. Palindrome & Vowels**

Write a program to check if a given string is palindrome and contains at least two different vowels.

Include a class UserMainCode with a static method checkPalindrome which accepts a string. The return type (integer) should be 1 if the above condition is satisfied, otherwise return -1.

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Note ? Case Insensitive while considering vowel, i.e a & A are same vowel, But Case sensitive while considering palindrome i.e abc CbA are not palindromes.

Input and Output Format:

Input consists of a string with maximum size of 100 characters.

Output consists of a single Integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

abceecba

**Sample Output 1:**

valid

**Sample Input 2:**

abcd

**Sample Output 2:**

invalid

**public** **class** UserMainCode {

**public** **static** **int** checkPlaindrome(String s1) {

                        StringBuffer sb = **new** StringBuffer(s1);

                        String s2 = sb.reverse().toString();

**if** (s1.equals(s2)) {

**return** 1;

                        } **else** {

**return** -1;

                        }

            }

}

**16. Strings ? Unique & Existing Characters**

Obtain two strings from user as input. Your program should modify the first string such that all the characters are replaced by plus sign (+) except the characters which are present in the second string.

That is, if one or more characters of first string appear in second string, they will not be replaced by +.

Return the modified string as output. Note - ignore case.

Include a class UserMainCode with a static method replacePlus which accepts two string variables. The return type is the modified string.

Create a Class Main which would be used to accept two Input strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two strings with maximum size of 100 characters.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

abcxyz

axdef

**Sample Output 1:**

a++ x++

**Sample Input 2:**

ABCDEF

feCBAd

**Sample Output 2:**

ABCDEF

**public** **class** UserMainCode {

**public** **static** String replacePlus(String s1, String s2) {

                        String output = s1;

**for** (**int** i = 0; i < s2.length(); i++) {

**char** c = s2.charAt(i);

               StringBuffer sb

                                    String temp = Character.*toString*(c);

**if** (output.contains(temp)) {

sb.append(temp)

else

sb.append(“+”)

                                                //output.replaceAll(temp, Character.*toString*('+'));

                                    }

                        }

**return** output;

            }

}

**17. PAN Card**

Write a program to read a string and validate PAN no. against following rules:

1. There must be eight characters.

2. First three letters must be alphabets followed by four digit number and ends with alphabet

3. All alphabets should be in capital case.

Print ?Valid? if the PAN no. is valid, else print ?Invalid?.

Include a class UserMainCode with a static method validatePAN which accepts a string. The return type (Integer) should return 1 if the string is a valid PAN no. else return 2.

Create a Class Main which would be used to accept a string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string, which corresponds to the PAN number.

Output consists of a string - "Valid" or "Invalid"

Refer sample output for formatting specifications.

**Sample Input 1:**

ALD3245E

**Sample Output 1:**

Valid

**Sample Input 2:**

OLE124F

**Sample Output 2:**

Invalid

**public** **class** UserMainCode {

**public** **static** **int** panCard(String s1) {

**if** (s1.matches("[A-Z]{3}[0-9]{4}[A-Z]{1}")) {

**return** 1;

                        } **else** {

**return** 2;

                        }

            }

}

**18. Longest Word**

Write a Program which finds the longest word from a sentence. Your program should read a sentence as input from user and return the longest word. In case there are two words of maximum length return the word which comes first in the sentence.

Include a class UserMainCode with a static method getLargestWord which accepts a string The return type is the longest word of type string.

Create a Class Main which would be used to accept two Input strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string with maximum size of 100 characters.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Welcome to the world of Programming

**Sample Output 1:**

Programming

**Sample Input 2:**

ABC DEF

**Sample Output 2:**

ABC

**public** **class** UserMainCode {

**public** **static** String longestWord(String s1) {

**int** max = 0;

                        String output = "";

                        String s[] = s1.split(" ");

**for** (**int** i = 0; i < s.length; i++) {

**if** (s[i].length() > max) {

                                                max = s[i].length();

                                                output = s[i];

                                    }

                        }

**return** output;

            }

}

**19. String Encryption**

Given an input as string and write code to encrypt the given string using following rules and return the encrypted string:

1. Replace the characters at odd positions by next character in alphabet.

2. Leave the characters at even positions unchanged.

Note:

- If an odd position charater is 'z' replace it by 'a'.

- Assume the first character in the string is at position 1.

Include a class UserMainCode with a static method encrypt which accepts a string.

The return type of the output is the encrypted string.

Create a Main class which gets string as an input and call the static method encrypt present in the UserMainCode.

Input and Output Format:

Input is a string .

Output is a string.

**Sample Input 1:**

curiosity

**Sample Output 1:**

dusipsjtz

**Sample Input 2:**

zzzz

**Sample Output 2:**

azaz

**public** **class** UserMainCode {

**public** **static** String encrypt(String s1) {

                        StringBuffer buffer = **new** StringBuffer();

**for** (**int** i = 0; i < s1.length(); i++) {

**if** (i % 2 == 0) {

**int** j = (**int**) s1.charAt(i);

**char** c = (**char**) (j + 1);

                                                buffer.append(c);

                                    } **else**

                                                buffer.append(s1.charAt(i));

                        }

**return** buffer.toString();

            }

}

**20. Fibonacci Sum**

Write a program to read an integer n, generate fibonacci series and calculate the sum of first n numbers in the series. Print the sum.

Include a class UserMainCode with a static method getSumOfNfibos which accepts an integer n. The return type (Integer) should return the sum of n fibonacci numbers.

Create a Class Main which would be used to accept an integer and call the static method present in UserMainCode.

Note: First two numbers in a Fibonacci series are 0, 1 and all other subsequent numbers are sum of its previous two numbers. Example - 0, 1, 1, 2, 3, 5...

Input and Output Format:

Input consists of an integer, which corresponds to n.

Output consists of an Integer (sum of fibonacci numbers).

Refer sample output for formatting specifications.

**Sample Input :**

5

**Sample Output :**

7

[0 + 1 + 1 + 2 + 3 = 7]

**public** **class** UserMainCode {

**public** **static** **int** getSumOfNFibos(**int** n) {

**int** first = 0, second = 1, next = 0, sum = 0;

                        ArrayList<Integer> list = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < n; i++) {

**if** (i == 0) {

                                                next = first;

                                    }

**if** (i == 1) {

                                                next = second;

                                    }

**if** (i > 1) {

                                                next = second + first;

                                                first = second;

                                                second = next;

                                    }

                                    list.add(next);

                        }

                        Iterator<Integer> iterator = list.iterator();

**while** (iterator.hasNext()) {

                                    Integer integer = iterator.next();

                                    sum = sum + integer;

                        }

**return** sum;

            }

}

**21. String Occurences**

Obtain two strings from user as input. Your program should count the number of occurences of second word of second sentence in the first sentence.

Return the count as output. Note - Consider case.

Include a class UserMainCode with a static method countNoOfWords which accepts two string variables. The return type is the modified string.

Create a Class Main which would be used to accept two Input strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two strings with maximum size of 100 characters.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

abc bcd abc bcd abc abc

av abc

**Sample Output 1:**

4

**Sample Input 2:**

ABC xyz AAA

w abc

**Sample Output 2:**

0

**public** **class** UserMainCode

{

**public** **static** **int** getValues (String s1, String s2)

            {

**int** count=0;

                        String []s3 = s1.split(" ");

                        String []s4 = s2.split(" ");

**for** (**int** i = 0; i < s3.length; i++)

                        {

**if**(s4[1].equals(s3[i]))

                                    {

                                                count++;

                                    }

                        }

**return** count;

            }

}

**22. Test Vowels**

Write a program to read a string and check if given string contains exactly five vowels in any order. Print ?Yes? if the condition satisfies, else print ?No?.

Assume there is no repetition of any vowel in the given string and all characters are lowercase.

Include a class UserMainCode with a static method testVowels which accepts a string. The return type (Integer) should return 1 if all vowels are present, else return 2.

Create a Class Main which would be used to accept a string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string (?Yes? or ?No?).

Refer sample output for formatting specifications.

**Sample Input 1:**

acbisouzze

**Sample Output 1:**

Yes

**Sample Input 2:**

cbisouzze

**Sample Output 2:**

No

**public** **class** UserMainCode {

**public** **static** **int** testVowels(String s) {

                        String s2 = s.replaceAll("[^aeiouAEIOU]", "");

                        HashSet<Character> hs = **new** HashSet<Character>();

**for** (**int** i = 0; i < s2.length(); i++) {

                                    hs.add(s2.charAt(i));

                        }

**if** (hs.size() == 5) {

**return** 1;

                        } **else** {

**return** 2;

                        }

            }

}

**23. Removing vowels from String**

Given a method with string input. Write code to remove vowels from even position in the string.

Include a class UserMainCode with a static method removeEvenVowels which accepts a string as input.

The return type of the output is string after removing all the vowels.

Create a Main class which gets string as an input and call the static method removeEvenVowels present in the UserMainCode.

Input and Output Format:

Input is a string .

Output is a string .

Assume the first character is at position 1 in the given string.

**Sample Input 1:**

commitment

**Sample Output 1:**

cmmitmnt

**Sample Input 2:**

capacity

**Sample Output 2:**

cpcty

**public** **class** UserMainCode {

**public** **static** String removeEvenVowels(String s1) {

                        StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < s1.length(); i++) {

**if** ((i % 2 == 0)) {

                                                sb.append(s1.charAt(i));

                                    } **else** **if** ((i % 2 != 0)) {

**if** (s1.charAt(i) != 'a' && s1.charAt(i) != 'e'

                                                                        && s1.charAt(i) != 'i' && s1.charAt(i) != 'o'

                                                                        && s1.charAt(i) != 'u') {

**if** (s1.charAt(i) != 'A' && s1.charAt(i) != 'E'

                                                                                    && s1.charAt(i) != 'I' && s1.charAt(i) != 'O'

                                                                                    && s1.charAt(i) != 'U') {

                                                                        sb.append(s1.charAt(i));

                                                            }

                                                }

                                    }

                        }

**return** sb.toString();

            }

}

**24. ArrayList Manipulation**

Write a program that performs the following actions:

Read 2n integers as input.

Create two arraylists to store n elements in each arraylist.

Write a function generateOddEvenList which accepts these two arraylist as input.

The function fetch the odd index elements from first array list and even index elements from second array list and add them to a new array list according to their index.

Return the arraylist.

Include a class UserMainCode with the static method generateOddEvenList which accepts two arraylist and returns an arraylist.

Create a Class Main which would be used to read 2n integers and call the static method present in UserMainCode.

**Note:**

- The index of first element is 0.

- Consider 0 as an even number.

- Maintain order in the output array list

Input and Output Format:

Input consists of 2n+1 integers. The first integer denotes the size of the arraylist, the next n integers are values to the first arraylist, and the last n integers are values to the second arraylist.

Output consists of a modified arraylist as per step 4.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

12

13

14

15

16

2

3

4

5

6

**Sample Output 1:**

2

13

4

15

6

**public** **class** Main {

**public** **static** **void** main(String[] args) {

                        Scanner sc = **new** Scanner(System.*in*);

**int** n = sc.nextInt();

                        ArrayList<Integer> al1 = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < n; i++) {

                                    al1.add(sc.nextInt());

                        }

                        ArrayList<Integer> al2 = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < n; i++) {

                                    al2.add(sc.nextInt());

                        }

                        ArrayList<Integer> al3 = **new** ArrayList<Integer>();

                        al3 = UserMainCode.*arrayListManipulation*(al1, al2);

                        Iterator<Integer> it = al3.iterator();

**while** (it.hasNext()) {

**int** res = it.next();

                                    System.*out*.println(res);

                        }

            }

}

**public** **class** UserMainCode {

**public** **static** ArrayList<Integer> arrayListManipulation(

                                    ArrayList<Integer> al1, ArrayList<Integer> al2) {

                        ArrayList<Integer> al3 = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < al1.size(); i++) {

**if** (i % 2 == 0) {

                                                al3.add(al2.get(i));

                                    } **else** {

                                                al3.add(al1.get(i));

                                    }

                        }

**return** al3;

            }

}

**25. Dash Check**

Write a program to read two strings and check whether or not they have dashes in the same places. Print ?Yes? if the condition satisfies, else print ?No?.

Include a class UserMainCode with a static method compareDashes which accepts two strings. The return type (Integer) should return 1 if all dashes are placed correctly, else return 2.

Create a Class Main which would be used to accept two strings and call the static method present in UserMainCode.

Note: The strings must have exactly the same number of dashes in exactly the same positions. The strings might be of different length.

Input and Output Format:

Input consists of two strings.

Output consists of a string (?Yes? or ?No?).

Refer sample output for formatting specifications.

**Sample Input 1:**

Hi-there-you.

12-(134)-7539

**Sample Output 1:**

Yes

**Sample Input 2:**

-15-389

-xyw-zzy

**Sample Output 2:**

No

**public** **static** **int** compareDashes(String s1, String s2)

            {

**int** count=0,k=0;

**int** a = s1.length();

**int** b = s2.length();

**int** l1[] = **new** **int**[a];

**int** l2[] = **new** **int**[b];

                        String st1[] = s1.split("-");

                        String st2[] = s2.split("-");

**for** ( k = 0; k < st1.length; k++)

                        {

                                    l1[k]=st1[k].length();

                        }

**for** (k = 0; k < st2.length; k++)

                        {

                                    l2[k]=st2[k].length();

                        }

**for** (**int** c = 0; c < k-1; c++)

                        {

**if**(l1[c]==l2[c])

                                    {

                                                count=1;

                                    }

**else**

                                    {

                                                count=0;

**break**;

                                    }

                        }

**return** count;

            }

}

**26. Sum of Powers of elements in an array**

Given a method with an int array. Write code to find the power of each individual element accoding to its position index, add them up and return as output.

Include a class UserMainCode with a static method getSumOfPower which accepts an integer array as input.

The return type of the output is an integer which is the sum powers of each element in the array.

Create a Main class which gets integer array as an input and call the static method getSumOfPower present in the UserMainCode.

**Input and Output Format:**

Input is an integer array.First element corresponds to the number(n) of elements in an array.The next inputs corresponds to each element in an array.

Output is an integer .

**Sample Input 1:**

4

3

6

2

1

**Sample Output 1:**

12

**Sample Input 2:**

4

5

3

7

2

**Sample Output 2:**

61

**public** **class** UserMainCode {

**public** **static** **int** sumPowerElements(**int** a[]) {

**int** sum = 0;

**for** (**int** i = 0; i < a.length; i++) {

                                    sum = (**int**) (sum + Math.*pow*(a[i], i));

                        }

**return** sum;

            }

}

**27. Duplicate Characters**

Write a Program which removes duplicate characters from the string. Your program should read a sentence (string) as input from user and return a string removing duplicate characters. Retain the first occurance of the duplicate character. Assume the characters are case ? sensitive.

Include a class UserMainCode with a static method removeDuplicates which accepts a string. The return type is the modified sentence of type string.

Create a Class Main which would be used to accept the input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string with maximum size of 100 characters.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

hi this is sample test

**Sample Output 1:**

hi tsample

**Sample Input 2:**

ABC DEF

**Sample Output 2:**

ABC DEF

**public** **class** UserMainCode {

**public** **static** String removeDuplicate(String s1) {

                        String res = **new** String("");

**for** (**int** i = 0; i < s1.length(); i++) {

**if** (!res.contains("" + s1.charAt(i))) {

                                                res += "" + s1.charAt(i);

                                    }

                        }

**return** res;

            }

}

**28. Reverse Split**

Write a program to read a string and a character, and reverse the string and convert it in a format such that each character is separated by the given character. Print the final string.

Include a class UserMainCode with a static method reshape which accepts a string and a character. The return type (String) should return the final string.

Create a Class Main which would be used to accept a string and a character, and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string and a character.

Output consists of a string (the final string).

Refer sample output for formatting specifications.

**Sample Input:**

Rabbit

-

**Sample Output:**

t-i-b-b-a-R

**public** **class** UserMainCode {

**public** **static** String reshape(String s1, **char** c) {

                        StringBuffer buffer = **new** StringBuffer(s1);

                        StringBuffer buffer2 = **new** StringBuffer();

                        buffer.reverse();

**for** (**int** i = 0; i < buffer.length(); i++) {

                                    buffer2.append(buffer.charAt(i)).append(c);

                        }

                        buffer2.deleteCharAt(buffer2.length() - 1);

**return** buffer2.toString();

            }

}

**29. Difference between largest and smallest elements in an array**

Given a method taking an int array having size more than or equal to 1 as input. Write code to return the difference between the largest and smallest elements in the array. If there is only one element in the array return the same element as output.

Include a class UserMainCode with a static method getBigDiff which accepts a integer array as input.

The return type of the output is an integer which is the difference between the largest and smallest elements in the array.

Create a Main class which gets integer array as an input and call the static method getBigDiff present in the UserMainCode.

Input and Output Format:

Input is an integer array.First element in the input represents the number of elements in an array.

Size of the array must be >=1

Output is an integer which is the difference between the largest and smallest element in an array

**Sample Input 1:**

4

3

6

2

1

**Sample Output 1:**

5

**Sample Input 2:**

4

5

3

7

2

**Sample Output 2:**

5

**public** **class** UserMainCode {

**public** **static** **int** largeSmallEle(**int** a[]) {

                        Arrays.*sort*(a);

**int** res = a[a.length - 1] - a[0];

**int** b = a.length;

**if** (b == 1) {

                                    res = a[0];

                        }

**return** res;

            }

}

**30. Find the element position in a reversed string array**

Given a method with an array of strings and one string variable as input. Write code to sort the given array in reverse alphabetical order and return the postion of the given string in the array.

Include a class UserMainCode with a static method getElementPosition which accepts an array of strings and a string variable as input.

The return type of the output is an integer which is the position of given string value from the array.

Create a Main class which gets string array and a string variable as an input and call the static method getElementPosition present in the UserMainCode.

Input and Output Format:

Input is an string array. First element in the input represents the size the arra

Assume the position of first element is 1.

Output is an integer which is the position of the string variable

**Sample Input 1:**

4

red

green

blue

ivory

ivory

**Sample Output 1:**

2

**Sample Input 2:**

3

grape

mango

apple

apple

Sample Output 2:

3

**public** **class** UserMainCode {

**public** **static** **int** elementPosInReversed(String s1[], String b) {

**int** output = 0;

                        ArrayList<String> al = **new** ArrayList<String>();

**for** (**int** i = 0; i < s1.length; i++) {

                                    al.add(s1[i]);

                        }

                        Collections.*sort*(al);

                        Collections.*reverse*(al);

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

**if** (b.equals(al.get(i))) {

                                                output = i + 1;

                                    }

                        }

**return** output;

            }

}

**31. Remove 10's**

Write a program to read an integer array and remove all 10s from the array, shift the other elements towards left and fill the trailing empty positions by 0 so that the modified array is of the same length of the given array.

Include a class UserMainCode with a static method removeTens which accepts the number of elements and an integer array. The return type (Integer array) should return the final array.

Create a Class Main which would be used to read the number of elements and the input array, and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of n+1 integers, where n corresponds to size of the array followed by n elements of the array.

Output consists of an integer array (the final array).

Refer sample output for formatting specifications.

**Sample Input :**

5

1

10

20

10

2

**Sample Output :**

1

20

2

o

o

**public** **class** UserMainCode {

**public** **static** **int**[] getDigitSum(**int** n, **int**[] a) {

                        ArrayList<Integer> arrayList = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < a.length; i++) {

                                    arrayList.add(a[i]);

                        }

                        ListIterator<Integer> iterator = arrayList.listIterator();

**while** (iterator.hasNext()) {

                                    Integer value = iterator.next();

**if** (value == 10) {

                                                iterator.remove();

                                                iterator.add(0);

                                    }

                        }

**int** res[] = **new** **int**[arrayList.size()];

**for** (**int** i = 0; i < res.length; i++) {

                                    res[i] = arrayList.get(i);

                        }

**return** res;

            }

}

**32. Mastering Hashmap**

You have recently learnt about hashmaps and in order to master it, you try and use it in all of your programs.

Your trainer / teacher has given you the following exercise:

Read 2n numbers as input where the first number represents a key and second one as value. Both the numbers are of type integers.

Write a function getAverageOfOdd to find out average of all values whose keys are represented by odd numbers. Assume the average is an int and never a decimal number. Return the average as output. Include this function in class UserMainCode.

Create a Class Main which would be used to read 2n numbers and build the hashmap. Call the static method present in UserMainCode.

Input and Output Format:

Input consists of a 2n+ 1 integers. The first integer specifies the value of n (essentially the hashmap size). The next pair of n numbers denote the key and value.

Output consists of an integer representing the average.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

2

34

1

4

5

12

4

22

**Sample Output 1:**

8

**public** **class** UserMainCode {

**public** **static** **int** getAverageOfOdd(HashMap<Integer, Integer> hm, **int** s) {

**int** sum = 0, avg = 0, count = 0;

                        Iterator<Integer> itr = hm.keySet().iterator();

                        {

**while** (itr.hasNext()) {

**int** j = itr.next();

**if** (j % 2 != 0) {

                                                            sum = sum + hm.get(j);

                                                            count++;

                                                }

                                    }

                                    avg = sum / count;

**return** avg;

                        }

            }

}

**33. Last Letters**

Write a program to read a sentence as a string and store only the last letter of each word of the sentence in capital letters separated by $. Print the final string.

Include a class UserMainCode with a static method getLastLetter which accepts a string. The return type (string) should return the final string.

Create a Class Main which would be used to read a string, and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string (the final string).

Refer sample output for formatting specifications.

**Smaple Input :**

This is a cat

**Sample Output :**

S$S$A$T

**public** **class** UserMainCode {

**public** **static** String lastLetter(String s1) {

                        String res = "";

                        String s[] = s1.split(" ");

**for** (**int** i = 0; i < s.length; i++) {

**if** (i < s.length - 1)

                                    res = res+ Character.*toUpperCase*(s[i].charAt(s[i].length() - 1))+ "$";

**else**

                                    res = res+ Character.*toUpperCase*(s[i].charAt(s[i].length() - 1));

                        }

**return** res;

            }

}

**34. Generate the series**

Given a method taking an odd positive Integer number as input. Write code to evaluate the following series:

1+3-5+7-9?+/-n.

Include a class UserMainCode with a static method addSeries which accepts a positive integer .

The return type of the output should be an integer .

Create a class Main which would get the input as a positive integer and call the static method addSeries present in the UserMainCode.

**Input and Output Format:**

Input consists of a positive integer n.

Output is a single integer .

Refer sample output for formatting specifications.

**Sample Input 1:**

9

**Sample Output 1:**

-3

**Sample Input 2:**

11

**Sample Output 2:**

8

**public** **class** UserMainCode {

**public** **static** **int** consectiveSum(**int** n) {

                        ArrayList<Integer> al1 = **new** ArrayList<Integer>();

**for** (**int** i = 1; i <= n; i++)

**if** (i % 2 != 0)

                                                al1.add(i);

**int** n1 = al1.get(0);

**for** (**int** i = 1; i < al1.size(); i++)

**if** (i % 2 != 0)

                                                n1 = n1 + al1.get(i);

**else**

                                                n1 = n1 - al1.get(i);

**return** n1;

            }

}

**35. Managers & Hashmaps**

A Company wants to automate its payroll process. You have been assigned as the programmer to build this package. You would like to showcase your skills by creating a quick prototype. The prototype consists of the following steps:

Read Employee details from the User. The details would include id, designation and salary in the given order. The datatype for id is integer, designation is string and salary is integer.

You decide to build two hashmaps. The first hashmap contains employee id as key and designation as value, and the second hashmap contains same employee ids as key and salary as value.

The company decides to hike the salary of managers by 5000. You decide to write a function increaseSalaries which takes the above hashmaps as input and returns a hashmap with only managers id and their increased salary as output. Include this function in class UserMainCode.

Create a Class Main which would be used to read employee details in step 1 and build the two hashmaps. Call the static method present in UserMainCode.

Input and Output Format:

Input consists of employee details. The first number indicates the size of the employees. The next three values indicate the employee id, employee designation and employee salary.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

2

2

programmer

3000

8

manager

50000

**Sample Output 1:**

8

55000

**public** **class** Main {

**public** **static** **void** main(String[] args) {

                        Scanner sc = **new** Scanner(System.*in*);

**int** s = Integer.*parseInt*(sc.nextLine());

                        HashMap<Integer, String> hm = **new** HashMap<Integer, String>();

                        HashMap<Integer, Integer> hm1 = **new** HashMap<Integer, Integer>();

**for** (**int** i = 0; i < s; i++) {

**int** id = Integer.*parseInt*(sc.nextLine());

                                    hm.put(id, sc.nextLine());

                                    hm1.put(id, Integer.*parseInt*(sc.nextLine()));

                        }

                        HashMap<Integer, Integer> hm2 = **new** HashMap<Integer, Integer>();

                        hm2 = UserMainCode.*managerHash*(hm, hm1);

                        Iterator<Integer> itr = hm2.keySet().iterator();

**while** (itr.hasNext()) {

**int** n = itr.next();

**int** fac = hm2.get(n);

                                    System.*out*.println(n);

                                    System.*out*.println(fac);

                        }

            }

}

**public** **class** UserMainCode {

**public** **static** HashMap<Integer, Integer> managerHash(

                                    HashMap<Integer, String> hm, HashMap<Integer, Integer> hm1) {

                        HashMap<Integer, Integer> hm3 = **new** HashMap<Integer, Integer>();

                        Iterator<Integer> itr = hm.keySet().iterator();

**while** (itr.hasNext()) {

**int** id = itr.next();

                                    String name = hm.get(id);

**if** (name.equals("manager")) {

**int** salary = hm1.get(id) + 5000;

                                                hm3.put(id, salary);

                                    }

                        }

**return** hm3;

            }

}

**36. Largest Key in HashMap**

Write a program that construts a hashmap and returns the value corresponding to the largest key.

Include a class UserMainCode with a static method getMaxKeyValue which accepts a string. The return type (String) should be the value corresponding to the largest key.

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of 2n+1 values. The first value corresponds to size of the hashmap. The next n pair of numbers equals the integer key and value as string.

Output consists of a string which is the value of largest key.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

12

amron

9

Exide

7

SF

**Sample Output 1:**

amron

**public** **class** UserMainCode {

**public** **static** String getValues(HashMap<Integer, String> hm) {

**int** b = 0, max = 0;

                        String s1 = **new** String();

                        Iterator<Integer> i = hm.keySet().iterator();

**while** (i.hasNext()) {

                                    b = i.next();

**if** (b > max) {

                                                max = b;

                                                s1 = hm.get(b);

                                    }

                        }

**return** s1;

            }

}

**37. Check first and last word**

Wrtte a program to check if the first word and the last word in the input string match.

Include a class UserMainCode with a static method ?check? that accepts a string argument and returns an int. If the first word and the last word in the string match, the method returns the number of characters in the word. Else the method returns the sum of the number of characters in the first word and last word.

Create a class Main which would get the input as a String and call the static method check present in the UserMainCode.

**Input and Output Format:**

Input consists of a string.

Output is an integer.

**Sample Input 1:**

how are you you are how

**Sample Output 1:**

3

**Sample Input 2:**

how is your child

**Sample Output 2:**

8

**public** **class** UserMainCode {

**public** **static** **int** checkFirstLast(String s1) {

**int** res = 0;

                        String s[] = s1.split(" ");

**if** (s[0].equals(s[s.length - 1])) {

                                    res = s[0].length();

                        } **else** {

                                    res = (s[0].length() + s[s.length - 1].length());

                        }

**return** res;

            }

}

**38. All Numbers**

Write a program to read a string array and return 1 if all the elements of the array are numbers, else return -1.

Include a class UserMainCode with a static method validateNumber which accepts a string aray. The return type (integer) should be -1 or 1 based on the above rules.

Create a Class Main which would be used to accept Input string array and call the static method present in UserMainCode.

The string array is said to be valid if all the elements in the array are numbers. Else it is invalid.

Input and Output Format:

Input consists of an integer specifying the size of string array followed by n strings.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

123

24.5

23

one

**Sample Output 1:**

invalid

**Sample Input 2:**

2

123

24.5

**Sample Output 2:**

valid

**public** **class** UserMainCode {

**public** **static** **int** validateNumbers(String s1[]) {

**int** output = 0;

**for** (**int** i = 0; i < s1.length; i++) {

**if** (s1[i].matches("[0-9]+[.]\*[0-9]\*")) {

                                                output = 1;

                                    } **else** {

                                                output = -1;

**break**;

                                    }

                        }

**return** output;

            }

}

**39. Concatenate Characters**

Given an array of Strings, write a program to take the last character of each string and make a new String by concatenating it.

Include a class UserMainCode with a static method ?concatCharacter? that accepts a String array as input and returns the new String.

Create a class Main which would get the String array as input and call the static method concatCharacter present in the UserMainCode.

Input and Output Format:

The first line of the input consists of an integer n that corresponds to the number of strings in the input string array.

The next n lines of the input consist of the strings in the input string array.

Output consists of a string.

**Sample Input:**

3

ab

a

abcd

**Sample Output:**

bad

**public** **class** UserMainCode {

**public** **static** String concatCharacter(String[] s) {

                        StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < s.length; i++) {

**int** l = s[i].length() - 1;

                                    sb.append(s[i].charAt(l));

                        }

**return** sb.toString();

            }

}

**40. Anagram**

Write a program to check whether the two given strings are anagrams.

Note: Rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once is called Anagram."

Include a class UserMainCode with a static method ?getAnagram? that accepts 2 strings as arguments and returns an int. The method returns 1 if the 2 strings are anagrams. Else it returns -1.

Create a class Main which would get 2 Strings as input and call the static method getAnagram present in the UserMainCode.

Input and Output Format:

Input consists of 2 strings. Assume that all characters in the string are lower case letters.

Output consists of a string that is either ?Anagrams? or ?Not Anagrams?.

**Sample Input 1:**

eleven plus two

twelve plus one

**Sample Output 1:**

Anagrams

**Sample Input 2:**

orchestra

carthorse

**Sample Output 2:**

Anagrams

**Sample Input 3:**

cognizant

technologies

**Sample Output 3:**

Not Anagrams

**public** **class** UserMainCode {

**public** **static** **int** getAnangram(String s1, String s2)

            {

**char**[] c1=s1.toCharArray();

**char**[] c2=s2.toCharArray();

                        Arrays.*sort*(c1);

                        Arrays.*sort*(c2);

                        String cc=**new** String(c1);

                        String cs=**new** String(c2);

**if**(cc.equals(cs))

                        {

**return** 1;

                        }

**else**

                        {

**return** -1;

                        }

            }

}

**41. Day of the Week**

Write a program to read a date as string (MM-dd-yyyy) and return the day of week on that date.

Include a class UserMainCode with a static method getDay which accepts the string. The return type (string) should be the day of the week.

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

07-13-2012

**Sample Output 1:**

Friday

**public** **class** UserMainCode {

**public** **static** String dayOfWeek(String s1) {

                        SimpleDateFormat sdf = **new** SimpleDateFormat("MM-dd-yyyy");

                        String day = **null**;

**try** {

                                    Date d = sdf.parse(s1);

                                    SimpleDateFormat sdf1 = **new** SimpleDateFormat("EEEEE");

                                    day = sdf1.format(d);

                        } **catch** (ParseException e) {

                                    e.printStackTrace();

                        }

**return** day;

            }

}

**42. Calculate Electricity Bill**

Given a method calculateElectricityBill() with three inputs. Write code to calculate the current bill.

Include a class UserMainCode with a static method calculateElectricityBill which accepts 3 inputs .The return type of the output should be an integer .

Create a class Main which would get the inputs and call the static method calculateElectricityBill present in the UserMainCode.

Input and Output Format:

Input consist of 3 integers.

First input is previous reading, second input is current reading and last input is per unit charge.

Reading Format - XXXXXAAAAA where XXXXX is consumer number and AAAAA is meter reading.

Output is a single integer corresponding to the current bill.

Refer sample output for formatting specifications.

**Sample Input 1:**

ABC2012345

ABC2012660

4

**Sample Output 1:**

1260

**Sample Input 2:**

ABCDE11111

ABCDE11222

3

**Sample Output 2:**

333

**public** **class** UserMainCode {

**public** **static** **int** electricityBill(String s1, String s2, **int** n) {

**int** n1 = Integer.*parseInt*(s1.substring(5, s1.length()));

**int** n2 = Integer.*parseInt*(s2.substring(5, s2.length()));

**int** res = Math.*abs*((n2 - n1) \* n);

**return** res;

            }

}

**43. Calculate Meter Reading**

Given 2 strings corresponding to the previous meter reading and the current meter reading, write a program to calculate electricity bill.

The input string is in the format ""AAAAAXXXXX"".

AAAAA is the meter code and XXXXX is the meter reading.

FORMULA: (XXXXX-XXXXX)\*4

Hint: if AAAAA of input1 and input2 are equal then separate the XXXXX from string and convert to integer. Assume that AAAAA of the 2 input strings will always be equal.

Include a class UserMainCode with a static method ?calculateMeterReading? that accepts 2 String arguments and returns an integer that corresponds to the electricity bill. The 1st argument corresponds to the previous meter reading and the 2nd arguement corresponds to the current meter reading.

Create a class Main which would get 2 Strings as input and call the static method calculateMeterReading present in the UserMainCode.

Input and Output Format:

Input consists of 2 strings. The first input corresponds to the previous meter reading and the second input corresponds to the current meter reading.

Output consists of an integer that corresponds to the electricity bill.

**Sample Input:**

CSECE12390

CSECE12400

**Sample Output:**

40

**public** **class** UserMainCode {

**public** **static** **int** calculateMeterReading(String s1, String s2) {

                        String newstr1 = s1.replaceAll("[^0-9]", "");

                        String newstr2 = s2.replaceAll("[^0-9]", "");

**int** num1, num2;

                        num1 = Integer.*parseInt*(newstr1);

                        num2 = Integer.*parseInt*(newstr2);

**int** op = (num2 - num1) \* 4;

**return** op;

            }

}

**44. Sum of Digits in a String**

Write code to get the sum of all the digits present in the given string.

Include a class UserMainCode with a static method sumOfDigits which accepts string input.

Return the sum as output. If there is no digit in the given string return -1 as output.

Create a class Main which would get the input and call the static method sumOfDigits present in the UserMainCode.

Input and Output Format:

Input consists of a string.

Output is a single integer which is the sum of digits in a given string.

Refer sample output for formatting specifications.

**Sample Input 1:**

good23bad4

**Sample Output 1:**

9

**Sample Input 2:**

good

**Sample Output 2:**

-1

**public** **class** UserMainCode {

**public** **static** **int** sumOfDigits(String s1) {

**int** sum = 0;

                        String s2 = s1.replaceAll("[a-zA-Z]", "");

**if** (s2.isEmpty()) {

**return** -1;

                        } **else** {

**int** a = Integer.*parseInt*(s2);

**while** (a != 0) {

**int** rem;

                                                rem = a % 10;

                                                sum += rem;

                                                a /= 10;

                                    }

                        }

**return** sum;

            }

}

**45. Max Substring**

Write a program to accept two string inputs. The first being a source string and second one a delimiter. The source string contains the delimiter at various locations. Your job is to return the substring with maximum number of characters. If two or more substrings have maximim number of characters return the substring which appears first. The size of the delimiter is 1.

Include a class UserMainCode with a static method extractMax which accepts the string. The return type (string) should be the max substring.

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a source string and delimiter.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

delhi-pune-patna

-

**Sample Output 1:**

delhi

**public** **class** UserMainCode {

**public** **static** String maxSubstring(String s1, String s4) {

**int** max = 0;

                        String s3 = **null**;

                        String[] s2 = s1.split(s4);

**for** (**int** i = 0; i < s2.length; i++) {

**int** len = s2[i].length();

**if** (len > max) {

                                                max = len;

                                                s3 = s2[i];

                                    }

                        }

**return** s3;

            }

}

**46. String Concatenation**

Write code to get two strings as input and If strings are of same length simply append them together and return the final string. If given strings are of different length, remove starting characters from the longer string so that both strings are of same length then append them together and return the final string.

Include a class UserMainCode with a static method concatstring which accepts two string input.

The return type of the output is a string which is the concatenated string.

Create a class Main which would get the input and call the static method concatstring present in the UserMainCode.

Input and Output Format:

Input consists of two strings.

Output is a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Hello

hi

**Sample Output 1:**

lohi

**Sample Input 2:**

Hello

Delhi

**Sample Output 2:**

HelloDelhi

**public** **class** UserMainCode {

**public** **static** String concatString(String s1, String s2) {

                        String res = "";

**int** a1 = s1.length();

**int** a2 = s2.length();

**if** (a1 == a2) {

                                    res = s1.concat(s2);

                        } **else** **if** (a1 > a2) {

                                    res = s1.substring(a1 - a2, a1);

                                    res = res.concat(s2);

                        } **else** **if** (a2 > a1) {

                                    res = s2.substring(a2 - a1, a2);

                                    res = s1.concat(res);

                        }

**return** res;

            }

}

**47. Retirement**

Given an input as HashMap which contains key as the ID and dob as value of employees, write a program to find out employees eligible for retirement. A person is eligible for retirement if his age is greater than or equal to 60.

Assume that the current date is 01/01/2014.

Include a class UserMainCode with a static method ?retirementEmployeeList? that accepts a HashMap as input and returns a ArrayList. In this method, add the Employee IDs of all the retirement eligible persons to list and return the sorted list.

(Assume date is in dd/MM/yyyy format).

Create a class Main which would get the HashMap as input and call the static method retirementEmployeeList present in the UserMainCode.

Input and Output Format:

The first line of the input consists of an integer n, that corresponds to the number of employees.

The next 2 lines of the input consists of strings that correspond to the id and dob of employee 1.

The next 2 lines of the input consists of strings that correspond to the id and dob of employee 2.

and so on...

Output consists of the list of employee ids eligible for retirement in sorted order.

**Sample Input :**

4

C1010

02/11/1987

C2020

15/02/1980

C3030

14/12/1952

T4040

20/02/1950

**Sample Output :**

[C3030, T4040]

**public** **class** UserMainCode {

**public** **static** ArrayList<String> hashmap(HashMap<String, String> hm) {

                        SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/yyyy");

                        ArrayList<String> al = **new** ArrayList<String>();

                        sdf.setLenient(**false**);

                        String s = "01/01/2014";

**try**

                        {

                                    Date d = **new** Date();

                                    d=sdf.parse(s);

                                    Calendar cal = Calendar.*getInstance*();

                                    cal.setTime(d);

                                    cal.add(Calendar.*YEAR*, -60);

                                    Iterator<String> itr = hm.keySet().iterator();

**while**(itr.hasNext())

                                    {

                                                String a = itr.next();

                                                String s1 = hm.get(a);

                                                Date d2 = **new** Date();

                                                d2=sdf.parse(s1);

                                                Calendar cal2 = Calendar.*getInstance*();

                                                cal2.setTime(d2);

**if**(cal2.before(cal))

                                                {

                                                            al.add(a);

                                                }

                                    }

                        }

**catch** (Exception e) {

                                    e.printStackTrace();

                        }

                        Collections.*sort*(al);

**return** al;

            }

**48. States and Capitals**

Write a program that construts a hashmap with ?state? as key and ?capital? as its value. If the next input is a state, then it should return capital$state in lowercase.

Include a class UserMainCode with a static method getCapital which accepts a hashmap. The return type is the string as given in the above statement

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of 2n+2 values. The first value corresponds to size of the hashmap. The next n pair of numbers contains the state and capital. The last value consists of the ?state? input.

Output consists of a string as mentioned in the problem statement.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

Karnataka

Bangaluru

Punjab

Chandigarh

Gujarat

Gandhinagar

Punjab

**Sample Output 1:**

chandigarh$punjab

**public** **class** UserMainCode {

**public** **static** String getCapital(HashMap<String, String> hm, String s1) {

                        String res = "";

**for** (Map.Entry e : hm.entrySet()) {

**if** (e.getKey().equals(s1)) {

                                                res = (e.getValue() + "$" + e.getKey());

**break**;

                                    }

                        }

**return** res;

            }

}

**49. Color Code**

Write a program to read a string and validate whether the given string is a valid color code based on the following rules:

- Must start with "#" symbol

- Must contain six characters after #

- It may contain alphabets from A-F or digits from 0-9

Include a class UserMainCode with a static method validateColorCode which accepts a string. The return type (integer) should return 1 if the color is as per the rules else return -1.

Create a Class Main which would be used to accept a String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string (Valid or Invalid).

Refer sample output for formatting specifications.

**Sample Input 1:**

#FF9922

**Sample Output 1:**

Valid

**Sample Input 2:**

#FF9(22

**Sample Output 2:**

Invalid

**public** **class** UserMainCode {

**public** **static** String getCapital(HashMap<String, String> hm, String s1) {

                        String res = "";

**for** (Map.Entry e : hm.entrySet()) {

**if** (e.getKey().equals(s1)) {

                                                res = (e.getValue() + "$" + e.getKey());

**break**;

                                    }

                        }

**return** res;

            }

}

**50. Kaprekar Number**

Write a program to check whether the given input number is a Kaprekar number or not.

Note : A positive whole number ?n? that has ?d? number of digits is squared and split into two pieces, a rig               ht-hand piece that has ?d? digits and a left-hand piece that has remaining ?d? or ?d-1? digits. If the sum of the two pieces is equal to the number, then ?n? is a Kaprekar number.

If its Kaprekar number assign to output variable 1 else -1.

Example 1:

Input1:9

9^2 = 81, right-hand piece of 81 = 1 and left hand piece of 81 = 8

Sum = 1 + 8 = 9, i.e. equal to the number. Hence, 9 is a Kaprekar number.

Example 2:

Input1:45

Hint:

45^2 = 2025, right-hand piece of 2025 = 25 and left hand piece of 2025 = 20

Sum = 25 + 20 = 45, i.e. equal to the number. Hence, 45 is a Kaprekar number."

Include a class UserMainCode with a static method ?getKaprekarNumber? that accepts an integer argument and returns an integer. The method returns 1 if the input integer is a Kaprekar number. Else the method returns -1.

Create a class Main which would get the an Integer as input and call the static method getKaprekarNumber present in the UserMainCode.

**Input and Output Format:**

Input consists of an integer.

Output consists of a single string that is either ?Kaprekar Number? or ?Not A Kaprekar Number?

**Sample Input 1:**

9

**Sample Output 1:**

Kaprekar Number

**Sample Input 2:**

45

**Sample Output 2:**

Kaprekar Number

**Sample Input 3:**

4

**Sample Output 3:**

Not A Kaprekar Number

**public** **class** UserMainCode {

**public** **static** **int** getKaprekarNumber(**int** a) {

**int** x = 0, y = 0, z = 0;

**int** b = a \* a;

                        String c = String.*valueOf*(b);

**int** d = c.length();

**if** (d == 2) {

                                    x = b % 10;

                                    y = b / 10;

                                    z = x + y;

                        } **else** **if** (d == 4) {

                                    x = b % 100;

                                    y = b / 100;

                                    z = x + y;

                        }

**if** (z == a) {

**return** 1;

                        } **else** {

**return** -1;

                        }

            }

}

**51. Simple String Manipulation – II(WordSum)**

Write a program to read a string and return an integer based on the following rules.

If the first word and the last word in the String match, then return the number of characters in the word else return sum of the characters in both words. Assume the Strings to be case - sensitive.

Include a class UserMainCode with a static method calculateWordSum which accepts a string. The return type (integer) should be based on the above rules.

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string with maximum size of 100 characters.

Output consists of a string.

Refer sample output for formatting specifications.

Sample Input 1:

COGNIZANT TECHNOLOGY SOLUTIONS COGNIZANT

**Sample Output 1:**

9

**Sample Input 2:**

HOW ARE YOU

**Sample Output 2:**

6

**public** **class** UserMainCode {

**public** **static** **int** getString(String s1) {

**int** res = 0;

                        String s[] = s1.split(" ");

**if** (s[0].equals(s[s.length - 1])) {

                                    res = s[0].length();

                        } **else** {

                                    res = (s[0].length() + s[s.length - 1].length());

                        }

**return** res;

            }

}

**52. Three Digits**

Write a program to read a string and check if the given string is in the format "CTS-XXX" where XXX is a three digit number.

Include a class UserMainCode with a static method validatestrings which accepts a string. The return type (integer) should return 1 if the string format is correct else return -1.

Create a Class Main which would be used to accept a String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string (Valid or Invalid).

Refer sample output for formatting specifications.

**Sample Input 1:**

CTS-215

**Sample Output 1:**

Valid

**Sample Input 2:**

CTS-2L5

**Sample Output 2:**

Invalid

**public** **class** UserMainCode {

**public** **static** **int** threeDigits(String s1) {

**if** (s1.matches("(CTS)[-]{1}[0-9]{3}")) {

**return** 1;

                        } **else** {

**return** -1;

                        }

            }

}

**53. Vowels**

Given a String input, write a program to find the word which has the the maximum number of vowels. If two or more words have the maximum number of vowels, print the first word.

Include a class UserMainCode with a static method ?storeMaxVowelWord? that accepts a string argument and returns the word containing the maximum number of vowels.

Create a class Main which would get the a String as input and call the static method storeMaxVowelWord present in the UserMainCode.

Input and Output Format:

Input consists of a string. The string may contain both lower case and upper case letters.

Output consists of a string.

**Sample Input :**

What is your name?

**Sample Output :**

your

**public** **class** UserMainCode {

**public** **static** String Vowels(String s1) {

**int** max = 0;

                        String s2 = "";

                        String s3 = "";

                        String s[] = s1.split(" ");

**for** (**int** i = 0; i < s.length; i++) {

                                    s2 = s[i].replaceAll("[^aeiouAEIOU]", "");

**int** len = s2.length();

**if** (len > max) {

                                                max = len;

                                                s3 = s[i];

                                    }

                        }

**return** s3;

            }

}

**54. Removing Keys from HashMap**

Given a method with a HashMap as input. Write code to remove all the entries having keys multiple of 4 and return the size of the final hashmap.

Include a class UserMainCode with a static method sizeOfResultandHashMap which accepts hashmap as input.

The return type of the output is an integer which is the size of the resultant hashmap.

Create a class Main which would get the input and call the static method sizeOfResultandHashMap present in the UserMainCode.

Input and Output Format:

First input corresponds to the size of the hashmap.

Input consists of a hashmap.

Output is an integer which is the size of the hashmap.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

2

hi

4

hello

12

hello world

**Sample Output 1:**

1

**Sample Input 2:**

3

2

hi

4

sdfsdf

3

asdf

**Sample Output 2:**

2

**public** **class** UserMainCode {

**public** **static** **int** removingKeys(HashMap<Integer, String> hm) {

**int** count = 0;

                        Iterator<Integer> itr = hm.keySet().iterator();

**while** (itr.hasNext()) {

**int** n = itr.next();

**if** (n % 4 != 0) {

                                                count++;

                                    }

                        }

**return** count;

            }

}

**55. Unique Characters**

Given a String as input , write a program to count and print the number of unique characters in it.

Include a class UserMainCode with a static method ?checkUnique? that accepts a String as input and returns the number of unique characters in it. If there are no unique characters in the string, the method returns -1.

Create a class Main which would get a String as input and call the static method checkUnique present in the UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of an integer.

**Sample Input 1:**

HOWAREYOU

**Sample Output 1:**

7

(Hint :Unique characters are : H,W,A,R,E,Y,U and other characters are repeating)

**Sample Input 2:**

MAMA

**Sample Output2:**

-1

**public** **class** UserMainCode {

**public** **static** **int** uniqueCounter(String s1) {

                        StringBuffer sb = **new** StringBuffer(s1);

**for** (**int** i = 0; i < sb.length(); i++) {

**int** count = 0;

**for** (**int** j = i + 1; j < sb.length(); j++) {

**if** (sb.charAt(i) == sb.charAt(j)) {

                                                            sb.deleteCharAt(j);

                                                            j--;

                                                            count++;

                                                }

                                    }

**if** (count >= 1) {

                                                sb.deleteCharAt(i);

                                                i--;

                                    }

                        }

**if** (sb.length() == 0) {

**return** -1;

                        } **else**

**return** sb.length();

            }

}

**56. Largest Element**

Write a program to read an int array of odd length, compare the first, middle and the last elements in the array and return the largest. If there is only one element in the array return the same element.

Include a class UserMainCode with a static method checkLargestAmongCorner which accepts an int arrayThe return type (integer) should return the largest element among the first, middle and the last elements.

Create a Class Main which would be used to accept Input array and call the static method present in UserMainCode.

Assume maximum length of array is 20.

Input and Output Format:

Input consists of n+1 integers. The first integer corresponds to n, the number of elements in the array. The next 'n' integers correspond to the elements in the array.

Output consists of a single Integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

2

3

8

4

5

**Sample Output 1:**

8

**public** **class** UserMainCode {

**public** **static** **int** checkLargestAmongCorner(**int** a[]) {

**int** s = a.length;

**int** res = 0;

**int** first = 0, mid = 0, last = 0;

**for** (**int** i = 0; i < s; i++) {

                                    first = a[0];

                                    mid = a[(s - 1) / 2];

                                    last = a[s - 1];

                        }

**if** (first > mid && first > last) {

                                    res = first;

                        } **else** **if** (mid > last && mid > first) {

                                    res = mid;

                        } **else** **if** (last > first && last > mid) {

                                    res = last;

                        }

**return** res;

            }

}

**57. Average of Primes**

Write a program to read an array and find average of all elements located at index i, where i is a prime number. Type cast the average to an int and return as output. The index starts from 0.

Include a class UserMainCode with a static method addPrimeIndex which accepts a single integer array. The return type (integer) should be the average of all elements located at index i where i is a prime number.

Create a Class Main which would be used to accept Input array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of n+1 integers. The first integer corresponds to n, the number of elements in the array. The next 'n' integers correspond to the elements in the array.

Output consists of a single Integer.

Refer sample output for formatting specifications.

Assume that the maximum number of elements in the array is 20 and minimum number of elements is 3.

**Sample Input 1:**

4

2

5

2

4

**Sample Output 1:**

3

**public** **class** UserMainCode {

**public** **static** **int** avgPrime(**int** a[]) {

**int** sum = 0, c = 0, j;

**for** (**int** i = 2; i < a.length; i++) {

**for** (j = 2; j < i; j++) {

**if** (i % j == 0) {

                                                            c++;

                                                }

                                    }

**if** (c == 0) {

                                                sum = sum + a[i];

                                    }

                        }

**int** avg = sum / 2;

**return** avg;

            }

}

**58. Vowels, Arrays & ArrayLists**

Write a program to read an array of strings and return an arraylist which consists of words whose both first and last characters are vowels. Assume all inputs are in lowecase.

Include a class UserMainCode with a static method matchCharacter which accepts a string array. The return type shoud be an arraylist which should contain elements as mentioned above.

Create a Class Main which would be used to accept Input array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of n+1 integers. The first integer corresponds to n, the number of elements in the array. The next 'n' string correspond to the elements in the array.

Output consists of strings which are elements of arraylist

Refer sample output for formatting specifications.

**Sample Input 1:**

4

abcde

pqrs

abci

orto

**Sample Output 1:**

abcde

abci

orto

**public** **class** UserMainCode {

**public** **static** ArrayList<String> vowelsArraysandArrayList(String[] str) {

            ArrayList<String> al = **new** ArrayList<String>();

**for** (**int** i = 0; i < str.length; i++) {

            String sp = str[i];

**char**[] mp = sp.toLowerCase().toCharArray();

**if** ((mp[0] == 'a' || mp[0] == 'e' || mp[0] == 'i' || mp[0] == 'o' || mp[0] == 'u')&& (mp[sp.length() - 1] == 'a'|| mp[sp.length() - 1] == 'e'|| mp[sp.length() - 1] == 'i'

|| mp[sp.length() - 1] == 'o' || mp[sp.length() - 1] == 'u'))

{

al.add(sp);

                                    }

                        }

**return** al;

            }

}

**59. nCr**

Write a program to calculate the ways in which r elements can be selected from n population, using nCr formula nCr=n!/r!(n-r)! where first input being n and second input being r.

Note1 : n! factorial can be achieved using given formula n!=nx(n-1)x(n-2)x..3x2x1.

Note2 : 0! = 1.

Example 5!=5x4x3x2x1=120

Include a class UserMainCode with a static method calculateNcr which accepts two integers. The return type (integer) should return the value of nCr.

Create a Class Main which would be used to accept Input elements and call the static method present in UserMainCode.

Input and Output Format:

Input consists of 2 integers. The first integer corresponds to n, the second integer corresponds to r.

Output consists of a single Integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

3

**Sample Output 1:**

4

**public** **class** UserMainCode {

**public** **static** **int** nCr(**int** n, **int** r) {

**int** fact = 1, fact1 = 1, fact2 = 1;

**for** (**int** i = 1; i <= n; i++) {

                                    fact = fact \* i;

                        }

**for** (**int** i = 1; i <= r; i++) {

                                    fact1 = fact1 \* i;

                        }

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

                                    fact2 = fact2 \* i;

                        }

**int** res = fact / (fact1 \* fact2);

**return** res;

            }

**60. ArrayList and Set Operations**

Write a program that performs the following actions:

Read 2n integers as input & a set operator (of type char).

Create two arraylists to store n elements in each arraylist.

Write a function performSetOperations which accepts these two arraylist and the set operator as input.

The function would perform the following set operations:.

'+' for SET-UNION

'\*' for SET-INTERSECTION

'-' for SET-DIFFERENCE

Refer to sample inputs for more details.

Return the resultant arraylist.

Include a class UserMainCode with the static method performSetOperations which accepts two arraylist and returns an arraylist.

Create a Class Main which would be used to read 2n+1 integers and call the static method present in UserMainCode.

Note:

- The index of first element is 0.

Input and Output Format:

Input consists of 2n+2 integers. The first integer denotes the size of the arraylist, the next n integers are values to the first arraylist, and the next n integers are values to the second arraylist and the last input corresponds to that set operation type.

Output consists of a modified arraylist as per step 4.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

1

2

3

3

5

7

+

**Sample Output 1:**

1

2

3

5

7

**Sample Input 2:**

4

10

9

8

7

2

4

6

8

\*

**Sample Output 2:**

8

**Sample Input 3:**

4

5

10

15

20

0

10

12

20

-

**Sample Output 3:**

5

15

**public** **class** UserMainCode {

**public** **static** **int**[] arrayListSetOp(ArrayList<Integer> a1,

                                    ArrayList<Integer> a2, **char** c) {

                        ArrayList<Integer> op1 = **new** ArrayList<Integer>();

**int** k = 0;

**switch** (c) {

**case** '+':

                                    a1.removeAll(a2);

                                    a1.addAll(a2);

                                    op1 = a1;

**break**;

**case** '\*':

                                    a1.retainAll(a2);

                                    op1 = a1;

**break**;

**case** '-':

**for** (**int** i = 0; i < a1.size(); i++) {

                                                k = 0;

**for** (**int** j = 0; j < a2.size(); j++) {

**if** (a1.get(i) == a2.get(j))

                                                                        k = 1;

                                                }

**if** (k == 0)

                                                            op1.add(a1.get(i));

                                    }

**break**;

                        }

                        // return op1;

**int** res[] = **new** **int**[op1.size()];

**for** (**int** i = 0; i < res.length; i++) {

                                    res[i] = op1.get(i);

                        }

**return** res;

            }

}

**61. Sum of Common Elements**

Write a program to find out sum of common elements in given two arrays. If no common elements are found print - ?No common elements?.

Include a class UserMainCode with a static method getSumOfIntersection which accepts two integer arrays and their sizes. The return type (integer) should return the sum of common elements.

Create a Class Main which would be used to accept 2 Input arrays and call the static method present in UserMainCode.

Input and Output Format:

Input consists of 2+m+n integers. The first integer corresponds to m (Size of the 1st array), the second integer corresponds to n (Size of the 2nd array), followed by m+n integers corresponding to the array elements.

Output consists of a single Integer corresponds to the sum of common elements or a string ?No common elements?.

Refer sample output for formatting specifications.

Assume the common element appears only once in each array.

**Sample Input 1:**

4

3

2

3

5

1

1

3

9

**Sample Output 1:**

4

**Sample Input 2:**

4

3

2

3

5

1

12

31

9

**Sample Output 2:**

No common elements

**public** **class** UserMainCode {

**public** **static** **int** getSumOfIntersection(**int** a[], **int** b[]) {

**int** sum = 0;

**for** (**int** i = 0; i < a.length; i++) {

**for** (**int** j = 0; j < b.length; j++) {

**if** (a[i] == b[j]) {

                                                            sum = sum + b[j];

                                                }

                                    }

                        }

**return** sum;

            }

}

**62. Transfer from Hashmap to Arraylist**

Write a program that constructs a hashmap with ?employee id? as key and ?name? as its value. Based on the rules below, on being satisfied, the name must be added to the arraylist.

i)First character should be small and the last character should be Capital.

ii)In name at least one digit should be there.

Include a class UserMainCode with a static method getName which accepts a hashmap. The return type is an arraylist as expected in the above statement

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of 2n+1 values. The first value corresponds to size of the hashmap. The next n pair of numbers contains the employee id and name.

Output consists of arraylist of strings as mentioned in the problem statement.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

1

ravi5raJ

2

sita8gitA

3

ram8sitA

4

rahul

**Sample Output 1:**

ravi5raJ

sita8gitA

ram8sitA

**public** **class** Main {

**public** **static** **void** main(String[] args) {

                        LinkedHashMap<Integer, String> hm1 = **new** LinkedHashMap<Integer, String>();

                        Scanner sc = **new** Scanner(System.*in*);

**int** n = Integer.*parseInt*(sc.nextLine());

**for** (**int** i = 0; i < n; i++) {

                                    hm1.put(Integer.*parseInt*(sc.nextLine()), sc.nextLine());

                        }

                        ArrayList<String> al1 = **new** ArrayList<String>();

                        al1 = UserMainCode.*getName*(hm1);

                        Iterator<String> it = al1.iterator();

**while** (it.hasNext()) {

                                    System.*out*.println(it.next());

                        }

            }

}

**public** **class** UserMainCode {

**public** **static** ArrayList<String> getName(LinkedHashMap<Integer, String> hm1) {

                        ArrayList<String> al2 = **new** ArrayList<String>();

                        Iterator<Integer> it = hm1.keySet().iterator();

**while** (it.hasNext()) {

**int** id = it.next();

                                    String name = hm1.get(id);

                                    String s1 = name.charAt(0) + "";

                                    String s2 = name.charAt(name.length() - 1) + "";

                                    String s3 = name.replaceAll("[a-zA-Z]", "");

**if** (s1.matches("[a-z]") && s2.matches("[A-z]") && s3.length() > 0) {

                                                al2.add(name);

                                    }

                        }

**return** al2;

            }

}

**63. Largest Span**

Write a program to read an array and find the size of largest span in the given array

""span"" is the number of elements between two repeated numbers including both numbers. An array with single element has a span of 1.

Include a class UserMainCode with a static method getMaxSpan which accepts a single integer array. The return type (integer) should be the size of largest span.

Create a Class Main which would be used to accept Input array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of n+1 integers. The first integer corresponds to n, the number of elements in the array. The next 'n' integers correspond to the elements in the array.

Output consists of a single Integer.

Refer sample output for formatting specifications.

Assume that the maximum number of elements in the array is 20.

**Sample Input 1:**

5

1

2

1

1

3

**Sample Output 1:**

4

**Sample Input 2:**

7

1

4

2

1

4

1

5

**Sample Output 2:**

6

**public** **class** UserMainCode {

**public** **static** **int** getLargestSpan(**int** a[]) {

**int** gap = 0, max = 0;

**for** (**int** i = 0; i < a.length; i++) {

**for** (**int** j = i + 1; j < a.length; j++) {

**if** (a[i] == a[j]) {

                                                            gap = j;

                                                }

                                    }

**if** (gap - i > max)

                                                max = gap - i;

                        }

**return** max + 1;

            }

}

**64. Validating Input Password**

102.Write a code get a password as string input and validate using the rules specified below. Apply following validations:

1. Minimum length should be 8 characters

2. Must contain any one of these three special characters @ or \_ or #

3. May contain numbers or alphabets.

4. Should not start with special character or number

5. Should not end with special character

Include a class UserMainCode with a static method validatePassword which accepts password string as input and returns an integer. The method returns 1 if the password is valid. Else it returns -1.

Create a class Main which would get the input and call the static method validatePassword present in the UserMainCode.

Input and Output Format:

Input consists of a string.

Output is a string Valid or Invalid.

Refer sample output for formatting specifications.

**Sample Input 1:**

ashok\_23

**Sample Output 1:**

Valid

**Sample Input 2:**

1980\_200

**Sample Output 2:**

Invalid

**public** **class** UserMainCode {

**public** **static** **boolean** validatePassword(String s1) {

**if** (s1.matches("[a-zA-Z0-9]+[\\@#\_\\?a-zA-Z0-9]+{8}")) {

**return** **true**;

                        } **else** {

**return** **false**;

                        }

            }

}

**65. Max Admissions**

Write a program that reads details about number of admissions per year of a particular college, return the year which had maximum admissions. The details are stored in an arraylist with the first index being year and next being admissions count.

Include a class UserMainCode with a static method getYear which accepts a arraylist. The return type is an integer indicating the year of max admissions.

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of 2n+1 values. The first value corresponds to size of the data (year & admissions). The next n pair of numbers contains the year and admissions count.

Output consists of an integer as mentioned in the problem statement.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

2010

200000

2011

300000

2012

45000

2013

25000

**Sample Output 1:**

2011

**public** **class** UserMainCode {

**public** **static** **int** getValues(HashMap<Integer, Integer> hm) {

**int** b = 0, max = 0, c = 0, d = 0;

                        Iterator<Integer> i = hm.keySet().iterator();

**while** (i.hasNext()) {

                                    b = i.next();

                                    c = hm.get(b);

**if** (c > max) {

                                                max = c;

                                                d = b;

                                    }

                        }

**return** d;

            }

}

**66. ID Validation**

Write a program to get two string inputs and validate the ID as per the specified format.

Include a class UserMainCode with a static method validateIDLocations which accepts two strings as input.

The return type of the output is a string Valid Id or Invalid Id.

Create a class Main which would get the input and call the static method validateIDLocations present in the UserMainCode.

Input and Output Format

Input consists of two strings.

First string is ID and second string is location. ID is in the format CTS-LLL-XXXX where LLL is the first three letters of given location and XXXX is a four digit number.

Output is a string Valid id or Invalid id.

Refer sample output for formatting specifications.

**Sample Input 1:**

CTS-hyd-1234

hyderabad

**Sample Output 1:**

Valid id

**Sample Input 2:**

CTS-hyd-123

hyderabad

**Sample Output 2:**

Invalid id

**public** **class** UserMainCode {

**public** **static** String validateIDLocation(String s1, String s2) {

                        String res = **null**;

                        String[] s3 = s1.split("-");

                        String id = s3[0];

                        String loc = s3[1];

                        String xxx = s3[2];

**if** (loc.matches(s2.substring(0, 3)) && xxx.length() == 4

                                                && id.matches("CTS")) {

                                    res = "Valid";

                        } **else** {

                                    res = "Invalid";

                        }

**return** res;

            }

}

**67. Max Scorer**

Write a program that performs the following actions:

Read n strings as input and stores them as an arraylist. The string consists of student information like name and obtained marks of three subjects. Eg: name-mark1-mark2-mark3 [suresh-70-47-12] The mark would range between 0 ? 100 (inclusive).

Write a function highestScorer which accepts these the arraylist and returns the name of the student who has scored the max marks. Assume the result will have only one student with max mark.

Include a class UserMainCode with the static method highestScorer which accepts the arraylist and returns the name (string) of max scorer.

Create a Class Main which would be used to read n strings into arraylist and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of 1 integer and n strings. The first integer denotes the size of the arraylist, the next n strings are score pattern described above.

Output consists of a string with the name of the top scorer.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

sunil-56-88-23

bindul-88-70-10

john-70-49-65

**Sample Output 1:**

john

**public** **class** UserMainCode {

**public** **static** String maxScorer(ArrayList<String> al, **int** n) {

                        String k = "";

                        String s1 = "";

**int** sum = 0, max = 0;

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

                                    k = al.get(i);

                                    String[] st = k.split("-");

**for** (**int** j = 0; j < k.length(); j++) {

                                                String s = st[0];

**int** a = Integer.*parseInt*(st[1]);

**int** b = Integer.*parseInt*(st[2]);

**int** c = Integer.*parseInt*(st[3]);

                                                sum = a + b + c;

**if** (sum > max) {

                                                            max = sum;

                                                            s1 = s;

                                                }

                                    }

                        }

**return** s1;

            }

}

**68. Sum Non Prime Numbers**

Write a program to calculate the sum of all the non prime positive numbers less than or equal to the given number.

Note: prime is a natural number greater than 1 that has no positive divisors other than 1 and itself

Example:

input = 9

Prime numbers = 2,3,5 and 7

output = 1+4+6+8+9=28

Include a class UserMainCode with a static method ?addNumbers? that accepts an integer arguement and returns an integer.

Create a class Main which would get an integer as input and call the static method validateNumber present in the UserMainCode.

**Input and Output Format:**

Input consists of an integer.

Output consists of an integer.

**Sample Input:**

9

**Sample Output:**

28

**public** **class** UserMainCode {

**public** **static** **int** func(**int** n) {

**int** sum = 0;

**int** k = 0;

**int** sum1 = 0;

**for** (**int** i = 2; i <= n; i++) {

                                    k = 0;

**for** (**int** j = 2; j < i; j++) {

**if** (i % j == 0)

                                                            k++;

                                    }

**if** (k == 0) {

                                                sum = sum + i;

                                    }

                        }

**for** (**int** r = 1; r <= n; r++)

                                    sum1 = sum1 + r;

**return** sum1 - sum;

            }

}

**69. Remove Elements**

Write a program to remove all the elements of the given length and return the size of the final array as output. If there is no element of the given length, return the size of the same array as output.

Include a class UserMainCode with a static method removeElements which accepts a string array, the number of elements in the array and an integer. The return type (integer) should return the size of the final array as output.

Create a Class Main which would be used to accept Input String array and a number and call the static method present in UserMainCode.

Assume maximum length of array is 20.

Input and Output Format:

Input consists of a integers that corresponds to n, followed by n strings and finally m which corresponds to the length value.

Output consists of a single Integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

a

bb

b

ccc

ddd

2

**Sample Output 1:**

4

**public** **class** UserMainCode {

**public** **static** **int** StringNotOfGivenLength(List<String> l1, **int** s1) {

**int** c = 0;

**for** (**int** i = 0; i < l1.size(); i++) {

**int** n2 = l1.get(i).length();

**if** (s1 != n2) {

                                                c++;

                                    }

                        }

**return** c;

            }

}

**70. Max Vowels**

Write a Program which fetches the word with maximum number of vowels. Your program should read a sentence as input from user and return the word with max number of vowels. In case there are two words of maximum length return the word which comes first in the sentence.

Include a class UserMainCode with a static method getWordWithMaximumVowels which accepts a string The return type is the longest word of type string.

Create a Class Main which would be used to accept two Input strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string with maximum size of 100 characters.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Appreciation is the best way to motivate

**Sample Output 1:**

Appreciation

**public** **class** UserMainCode {

**public** **static** String maxVowels(String s1) {

**int** max = 0;

                        String s2 = "";

                        String s3 = "";

                        String s[] = s1.split(" ");

**for** (**int** i = 0; i < s.length; i++) {

                                    s2 = s[i].replaceAll("[^aeiouAEIOU]", "");

**int** len = s2.length();

**if** (len > max) {

                                                max = len;

                                                s3 = s[i];

                                    }

                        }

**return** s3;

            }

}

**71. Find the difference between Dates in months**

Given a method with two date strings in yyyy-mm-dd format as input. Write code to find the difference between two dates in months.

Include a class UserMainCode with a static method getMonthDifference which accepts two date strings as input.

The return type of the output is an integer which returns the diffenece between two dates in months.

Create a class Main which would get the input and call the static method getMonthDifference present in the UserMainCode.

Input and Output Format:

Input consists of two date strings.

Format of date : yyyy-mm-dd.

Output is an integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

2012-03-01

2012-04-16

**Sample Output 1:**

1

**Sample Input 2:**

2011-03-01

2012-04-16

**Sample Output 2:**

13

**public** **class** UserMainCode {

**public** **static** **int** date(String s1, String s2) {

**int** res = 0;

                        SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd");

                        sdf.setLenient(**false**);

**try** {

                                    Date d = sdf.parse(s1);

                                    Date d1 = sdf.parse(s2);

                                    Calendar cal = Calendar.*getInstance*();

                                    cal.setTime(d);

**int** m1 = cal.get(Calendar.*MONTH*);

**int** y1 = cal.get(Calendar.*YEAR*);

                                    cal.setTime(d1);

**int** m2 = cal.get(Calendar.*MONTH*);

**int** y2 = cal.get(Calendar.*YEAR*);

**if** (y1 > y2) {

                                                y1 -= 1;

                                    } **else** {

                                                y2 -= 1;

                                    }

**if** (m1 > m2) {

                                                m1 = 12 - m1;

                                    } **else** {

                                                m2 = 12 - m2;

                                    }

                                    res = Math.*abs*(y2 - y1) \* 12 + Math.*abs*(m2 + m1);

                        } **catch** (ParseException e) {

                                    e.printStackTrace();

                        }

**return** res;

            }

}

**72. Date Format Conversion**

Given a date string in the format dd/mm/yyyy, write a program to convert the given date to the format dd-mm-yy.

Include a class UserMainCode with a static method ?convertDateFormat? that accepts a String and returns a String.

Create a class Main which would get a String as input and call the static method convertDateFormat present in the UserMainCode.

Input and Output Format:

Input consists of a String.

Output consists of a String.

**Sample Input:**

12/11/1998

**Sample Output:**

12-11-98

**public** **class** UserMainCode {

**public** **static** String getValues(String s1) {

                        SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/yyyy");

                        sdf.setLenient(**false**);

                        String s2 = "";

**try** {

                                    Date d1 = sdf.parse(s1);

                                    SimpleDateFormat sdf1 = **new** SimpleDateFormat("dd-MM-yy");

                                    s2 = sdf1.format(d1);

                        } **catch** (Exception e) {

                                    e.printStackTrace();

                        }

**return** s2;

            }

}

**73. All Vowels**

Write a Program to check if given word contains exactly five vowels and the vowels are in alphabetical order. Return 1 if the condition is satisfied else return -1. Assume there is no repetition of any vowel in the given string and all letters are in lower case.

Include a class UserMainCode with a static method testOrderVowels which accepts a string The return type is integer based on the condition stated above.

Create a Class Main which would be used to accept two Input strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string with maximum size of 100 characters.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

acebisouzz

**Sample Output 1:**

valid

**Sample Input 2:**

alphabet

**Sample Output 2:**

invalid

**public** **static** **int** getValues(String s1) {

                        String s2 = "aeiou";

                        StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < s1.length(); i++) {

**for** (**int** j = 0; j < s2.length(); j++) {

**if** (s1.charAt(i) == s2.charAt(j)) {

                                                            sb.append(s1.charAt(i));

                                                }

                                    }

                        }

**if** (sb.toString().equals(s2)) {

**return** 1;

                        } **else** {

**return** -1;

                        }

            }

}

**74. Sum of cubes and squares of elements in an array**

Write a program to get an int array as input and identify even and odd numbers. If number is odd get cube of it, if number is even get square of it. Finally add all cubes and squares together and return it as output.

Include a class UserMainCode with a static method addEvenOdd which accepts integer array as input.

The return type of the output is an integer which is the sum of cubes and squares of elements in the array.

Create a class Main which would get the input and call the static method addEvenOdd present in the UserMainCode.

Input and Output Format

Input consists of integer array.

Output is an integer sum.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

2

6

3

4

5

**Sample Output 1:**

208

**public** **class** UserMainCode {

**public** **static** **int** addEvenOdd(**int** a[]) {

**int** cubesum = 0, squaresum = 0;

**for** (**int** i = 0; i < a.length; i++) {

**if** (a[i] % 2 != 0) {

                                                cubesum += a[i] \* a[i] \* a[i];

                                    } **else** {

                                                squaresum += a[i] \* a[i];

                                    }

                        }

**int** cs = cubesum + squaresum;

**return** cs;

            }

}

**75. Valid Date**

Given a date string as input, write a program to validate if the given date is in any of the following formats:

dd.mm.yyyy

dd/mm/yy

dd-mm-yyyy

Include a class UserMainCode with a static method ?validateDate? that accepts a String and returns an integer. This method returns 1 if the date is valid, else return -1.

Create a class Main which would get a String as input and call the static method validateDate present in the UserMainCode.

Input and Output Format:

Input consists of a String.

Output consists of a String that is either 'Valid' or 'Invalid'.

**Sample Input 1:**

12.03.2012

**Sample Output 1:**

Valid

Sample Input 2:

27#01#1977

**Sample Output 2:**

Invalid

**public** **class** UserMainCode

{

**public** **static** **int** validateDate(String s)

            {

**int** result=-1;

**if**(s.matches("[0-9]{2}[.]{1}[0-9]{2}[.]{1}[0-9]{4}"))

                        {

                                    result=1;

                        }

**else** **if**(s.matches("[0-9]{2}[/]{1}[0-9]{2}[/]{1}[0-9]{4}"))

                        {

                                    result=1;

                        }

**else** **if**(s.matches("[0-9]{2}[-]{1}[0-9]{2}[-]{1}[0-9]{4}"))

                        {

                                    result=1;

                        }

**return** result;

            }

}

**76. Adjacent Swaps**

Write a Program that accepts a string as a parameter and returns the string with each pair of adjacent letters reversed. If the string has an odd number of letters, the last letter is unchanged.

Include a class UserMainCode with a static method swapPairs which accepts a string. The return type is string which is reversed pair of letters.

Create a Class Main which would be used to accept two Input strings and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string with maximum size of 100 characters.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

forget

**Sample Output 1:**

ofgrte

**Sample Input 2:**

New York

**Sample Output 2:**

eN woYkr

**public** **class** UserMainCode {

**public** **static** String getValues(String s1) {

                        StringBuffer sb = **new** StringBuffer();

**int** l = s1.length();

**if** (l % 2 == 0) {

**for** (**int** i = 0; i < s1.length() - 1; i = i + 2) {

**char** a = s1.charAt(i);

**char** b = s1.charAt(i + 1);

                                                sb.append(b).append(a);

                                    }

**return** sb.toString();

                        } **else** {

**for** (**int** i = 0; i <= s1.length()-3;i=i+2) {

**char** a = s1.charAt(i);

**char** b = s1.charAt(i + 1);

                                                sb.append(b).append(a);

                                    }

                                    sb.append(s1.charAt(l - 1));

**return** sb.toString();

                        }

            }

}

**77. Convert Format**

Given a 10 digit positive number in the format XXX-XXX-XXXX as a string input, write a program to convert this number to the format XX-XX-XXX-XXX.

Include a class UserMainCode with a static method ?convertFormat? that accepts a String argument and returns a String.

Create a class Main which would get a String as input and call the static method convertFormat present in the UserMainCode.

Input and Output Format:

Input consists of a String.

Output consists of a String.

**Sample Input:**

555-666-1234

**Sample Output:**

55-56-661-234

**public** **class** UserMainCode {

**public** **static** String getValues(String s) {

                        String st[] = s.split("-");

                        String s1 = st[0];

                        String s2 = st[1];

                        String s3 = st[2];

                        StringBuffer sb = **new** StringBuffer();

                        sb.append(s1.substring(0,s1.length()-1)).append('-');

                        sb.append(s1.charAt(s1.length()-1)).append(s2.charAt(0)).append('-');

                        sb.append(s2.substring(1, s2.length())).append(s3.charAt(0)).append('-');

                        sb.append(s3.substring(1, s3.length()));

**return** sb.toString();

            }

}

**78. IP Validator**

Write a program to read a string and validate the IP address. Print ?Valid? if the IP address is valid, else print ?Invalid?.

Include a class UserMainCode with a static method ipValidator which accepts a string. The return type (integer) should return 1 if it is a valid IP address else return 2.

Create a Class Main which would be used to accept Input String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string that corresponds to an IP.

Output consists of a string(?Valid? or ?Invalid?).

Refer sample output for formatting specifications.

Note: An IP address has the format a.b.c.d where a,b,c,d are numbers between 0-255.

**Sample Input 1:**

132.145.184.210

**Sample Output 1:**

Valid

**Sample Input 2:**

132.145.184.290

**Sample Output 2:**

Invalid

**public** **class** UserMainCode {

**public** **static** **int** ipValidator(String s1) {

**int** flag = 0;

                        String[] st = s1.split("\\.");

**for** (**int** i = 0; i < st.length; i++) {

                                    String s2 = st[i];

**int** n = Integer.*parseInt*(s2);

**if** (n > 0 && n <= 255) {

                                                flag++;

                                    }

                        }

**if** (flag == 4) {

**return** 1;

                        } **else** {

**return** 2;

                        }

            }

}

**79. Sum of Digits**

Write a Program that accepts a word as a parameter, extracts the digits within the string and returns its sum.

Include a class UserMainCode with a static method getdigits which accepts a string. The return type is integer representing the sum.

Create a Class Main which would be used to accept the input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string with maximum size of 100 characters.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

abc12de4

**Sample Output 1:**

7

**public** **class** UserMainCode {

**public** **static** **int** sumOfDigits(String s1) {

**int** sum = 0;

                        String s2 = s1.replaceAll("[a-zA-Z]", "");

**if** (s2.isEmpty()) {

**return** -1;

                        } **else** {

**int** a = Integer.*parseInt*(s2);

**while** (a != 0) {

**int** rem;

                                                rem = a % 10;

                                                sum += rem;

                                                a /= 10;

                                    }

                        }

**return** sum;

            }

}

**80. Add and Reverse**

Given an int array and a number as input, write a program to add all the elements in the array greater than the given number. Finally reverse the digits of the obtained sum and print it.

Include a class UserMainCode with a static method ?addAndReverse? that accepts 2 arguments and returns an integer.The first argument corresponds to the integer array and the second argument corresponds to the number.

Create a class Main which would get the required input and call the static method addAndReverse present in the UserMainCode.

Example:

Input Array = {10,15,20,25,30,100}

Number = 15

sum = 20 + 25 + 30 + 100 = 175

output = 571

**Input and Output Format:**

The first line of the input consists of an integer that corresponds to the number of elements in the array.

The next n lines of the input consists of integers that correspond to the elements in the array.

The last line of the input consists of an integer that corresponds to the number.

Output consists of a single integer.

**Sample Input**

6

10

15

20

25

30

100

15

**Sample Output**

571

**public** **class** UserMainCode {

**public** **static** **int** addAndReverse(**int** a[], **int** n) {

**int** sum = 0;

**int** ss = a.length;

**for** (**int** i = ss - 1; i >= 0; i--) {

**if** (a[i] <= n) {

**break**;

                                    } **else** {

                                                sum += a[i];

                                    }

                        }

                        StringBuilder sb = **new** StringBuilder(String.*valueOf*(sum));

                        sb.reverse();

**return** Integer.*valueOf*(sb.toString());

            }

}

**81. Password**

Given a String , write a program to find whether it is a valid password or not.

Validation Rule:

Atleast 8 characters

Atleast 1 number(1,2,3...)

Atleast 1 special character(@,#,%...)

Atleast 1 alphabet(a,B...)

Include a class UserMainCode with a static method ?validatePassword? that accepts a String argument and returns a boolean value. The method returns true if the password is acceptable. Else the method returns false.

Create a class Main which would get a String as input and call the static method validatePassword present in the UserMainCode.

Input and Output Format:

Input consists of a String.

Output consists of a String that is either ?Valid? or ?Invalid?.

**Sample Input 1:**

cts@1010

**Sample Output 1:**

Valid

**Sample Input 2:**

punitha3

**Sample Output 2:**

Invalid

**public** **class** UserMainCode {

**public** **static** **boolean** validatePassword(String s) {

**if** (s.matches("((?=.\*[0-9])(?=.\*[a-zA-Z])(?=.\*[@#$!]).{8,})")) {

**return** **true**;

                        } **else** {

**return** **false**;

                        }

            }

}

**82. Difference between two dates in days**

Get two date strings as input and write code to find difference between two dates in days.

Include a class UserMainCode with a static method getDateDifference which accepts two date strings as input.

The return type of the output is an integer which returns the diffenece between two dates in days.

Create a class Main which would get the input and call the static method getDateDifference present in the UserMainCode.

Input and Output Format:

Input consists of two date strings.

Format of date : yyyy-mm-dd.

Output is an integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

2012-03-12

2012-03-14

**Sample Output 1:**

2

**Sample Input 2:**

2012-04-25

2012-04-28

**Sample Output 2:**

3

**public** **class** UserMainCode {

**public** **static** **int** Datediff(String s1, String s2) {

**int** n = 0;

                        SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd");

**try** {

                                    Date d = sdf.parse(s1);

                                    Date d1 = sdf.parse(s2);

**double** diff = Math.*abs*(d.getTime() - d1.getTime());

**double** factor = 1000 \* 3600 \* 24;

**int** difference = (**int**) (diff / factor);

                                    n = difference;

                        } **catch** (Exception e) {

                                    System.*out*.println("Enter date in correct format");

                        }

**return** n;

            }

}

**83. Next Year day**

Given a date string in dd/mm/yyyy format, write a program to calculate the day which falls on the same date next year. Print the output in small case.

The days are sunday, monday, tuesday, wednesday, thursday, friday and saturday.

Include a class UserMainCode with a static method ?nextYearDay? that accepts a String and returns a String.

Create a class Main which would get a String as input and call the static method nextYearDay present in the UserMainCode.

Input and Output Format:

Input consists of a String.

Output consists of a String.

**Sample Input:**

13/07/2012

**Sample Output:**

saturday

**public** **class** UserMainCode {

**public** **static** String nextYearDay(String s1) {

                        String s = "";

                        SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/yyyy");

                        sdf.setLenient(**false**);

**try** {

                                    Date date = sdf.parse(s1);

                                    Calendar cal = Calendar.*getInstance*();

                                    cal.setTime(date);

                                    cal.add(Calendar.*YEAR*, 1);

                                    Date date2 = cal.getTime();

                                    SimpleDateFormat sdf1 = **new** SimpleDateFormat("EEEEE");

                                    s = sdf1.format(date2);

                        } **catch** (Exception e) {

                                    e.printStackTrace();

                        }

**return** s;

            }

}

**84. Employee Bonus**

A Company wants to give away bonus to its employees. You have been assigned as the programmer to automate this process. You would like to showcase your skills by creating a quick prototype. The prototype consists of the following steps:

Read Employee details from the User. The details would include id, DOB (date of birth) and salary in the given order. The datatype for id is integer, DOB is string and salary is integer.

You decide to build two hashmaps. The first hashmap contains employee id as key and DOB as value, and the second hashmap contains same employee ids as key and salary as value.

If the age of the employee in the range of 25 to 30 years (inclusive), the employee should get bonus of 20% of his salary and in the range of 31 to 60 years (inclusive) should get 30% of his salary. store the result in TreeMap in which Employee ID as key and revised salary as value. Assume the age is caculated based on the date 01-09-2014. (Typecast the bonus to integer).

Other Rules:

a. If Salary is less than 5000 store -100.

b. If the age is less than 25 or greater than 60 store -200.

c. a takes more priority than b i.e both if a and b are true then store -100.

You decide to write a function calculateRevisedSalary which takes the above hashmaps as input and returns the treemap as output. Include this function in class UserMainCode.

Create a Class Main which would be used to read employee details in step 1 and build the two hashmaps. Call the static method present in UserMainCode.

Input and Output Format:

Input consists of employee details. The first number indicates the size of the employees. The next three values indicate the employee id, employee DOB and employee salary. The Employee DOB format is ?dd-mm-yyyy?

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

2

1010

20-12-1987

10000

2020

01-01-1985

14400

**Sample Output 1:**

1010

12000

2020

17280

**85. File Extension**

Write a program to read a file name as a string and find out the file extension and return it as output. For example, the file sun.gif has the extension gif.

Include a class UserMainCode with a static method fileIdentifier which accepts a string. The return type (string) should return the extension of the input string (filename).

Create a Class Main which would be used to accept Input String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string that corresponds to a file name.

Output consists of a string(extension of the input string (filename)).

Refer sample output for formatting specifications.

**Sample Input 1:**

sun.gif

**Sample Output 1:**

gif

**public** **class** UserMainCode {

**public** **static** String extension(String s1) {

                        String s[] = s1.split("\\.");

**return** s[1];

                        }

}

**86. Sum Squares of Digits**

Write a program that accepts a positive number as input and calculates the sum of squares of individual digits of the given number.

Include a class UserMainCode with a static method ?getSumOfSquaresOfDigits? that accepts an integer argument and returns an integer.

Create a class Main which would get an integer as input and call the static method getSumOfSquaresOfDigits present in the UserMainCode.

Input and Output Format:

Input consists of an integer.

Output consists of an integer.

**Sample Input:**

321

**Sample Output:**

14

**public** **class** UserMainCode {

**public** **static** **int** getSumofSq(**int** a) {

**int** rem, num = 1, sum = 0;

**while** (a != 0) {

                                    rem = a % 10;

                                    num = rem \* rem;

                                    sum += num;

                                    a /= 10;

                        }

**return** sum;

            }

}

**87. Grade Calculator**

A School wants to assign grades to its students based on their marks. You have been assigned as the programmer to automate this process. You would like to showcase your skills by creating a quick prototype. The prototype consists of the following steps:

Read student details from the User. The details would include roll no, mark in the given order. The datatype for id is integer, mark is integer.

You decide to build a hashmap. The hashmap contains roll no as key and mark as value.

**BUSINESS RULE:**

1. If Mark is greater than or equal to 80 store medal as ""GOLD"".

2. If Mark is less then to 80 and greater than or equal to 60 store medal as ""SILVER"".

3 .If Mark is less then to 60 and greater than or equal to 45 store medal as ""BRONZE"" else store ""FAIL"".

Store the result in TreeMap in which Roll No as Key and grade as value.

4. You decide to write a function calculateGrade which takes the above hashmaps as input and returns the treemap as output. Include this function in class UserMainCode.

Create a Class Main which would be used to read employee details in step 1 and build the two hashmaps. Call the static method present in UserMainCode.

Input and Output Format:

Input consists of employee details. The first number indicates the size of the students. The next two values indicate the roll id, mark.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

2

1010

80

100

40

**Sample Output 1:**

100

FAIL

1010

GOLD

**public** **class** UserMainCode {

**public** **static** TreeMap<Integer, String> examResult(

                                    HashMap<Integer, Integer> m1) {

                        TreeMap<Integer, String> t1 = **new** TreeMap<Integer, String>();

**int** s1 = 0, n = 0;

                        Iterator<Integer> i = m1.keySet().iterator();

**while** (i.hasNext()) {

                                    n = i.next();

                                    s1 = m1.get(n);

**if** (s1 >= 80) {

                                                t1.put(n, "GOLD");

                                    } **else** **if** (s1 < 80 && s1 >= 60) {

                                                t1.put(n, "SILVER");

                                    } **else** **if** (s1 < 60 && s1 >= 45) {

                                                t1.put(n, "BRONZE");

                                    } **else** {

                                                t1.put(n, "FAIL");

                                    }

                        }

**return** t1;

            }

}

**88. Even and Odd Index Sum**

Write a program that accepts a positive number as input and calculates the sum of digits at even indexes (say evenSum) and sum of digits at odd indexes (say oddSum) in the given number. If both the sums are equal , print 'yes', else print no.

Example:

input = 23050

evenSum = 2 + 0 + 0 = 2

oddSum = 3 + 5 = 8

output = no

Include a class UserMainCode with a static method ?sumOfOddEvenPositioned? that accepts an integer and returns an integer. The method returns 1 if the 2 sums are equal. Else the method returns -1.

Create a class Main which would get an integer as input and call the static method sumOfOddEvenPositioned present in the UserMainCode.

Input and Output Format:

Input consists of an integer.

Output consists of a string that is either ?yes? or ?no?.

**Sample Input 1:**

23050

**Sample Output 1:**

no

**Sample Input 2:**

231

**Sample Output 2:**

yes

**public** **class** UserMainCode {

**public** **static** **int** evenOddSum(**int** a) {

**int** rem, count = 0, esum = 0, osum = 0;

**while** (a != 0) {

                                    rem = a % 10;

**if** (count % 2 == 0) {

                                                esum = esum + rem;

                                    } **else** {

                                                osum = osum + rem;

                                    }

                                    count++;

                                    a = a / 10;

                        }

**if** (esum == osum) {

**return** 1;

                        } **else** {

**return** -1;

                        }

            }

}

**89. Digits - II**

Write a program to read a non-negative integer n, compute the sum of its digits. If sum is greater than 9 repeat the process and calculate the sum once again until the final sum comes to single digit.Return the single digit.

Include a class UserMainCode with a static method getDigitSum which accepts the integer value. The return type is integer.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a integer.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

9999

**Sample Output 1:**

9

**Sample Input 2:**

698

**Sample Output 2:**

5

**public** **class** UserMainCode {

**public** **static** **int** getDigitSum(**int** digit) {

**int** rem, sum = 0, dsum = 0, rem1;

**while** (digit != 0) {

                                    rem = digit % 10;

                                    sum = sum + rem;

                                    digit /= 10;

                        }

**if** (sum < 9) {

                                    System.*out*.println(sum);

                        } **else** {

**while** (sum != 0) {

                                                rem1 = sum % 10;

                                                dsum += rem1;

                                                sum /= 10;

                                    }

                        }

**return** dsum;

            }

}

**90. Find common characters and unique characters in string**

Given a method with two strings as input. Write code to count the common and unique letters in the two strings.

Note:

- Space should not be counted as a letter.

- Consider letters to be case sensitive. ie, "a" is not equal to "A".

Include a class UserMainCode with a static method commonChars which accepts two strings as input.

The return type of the output is the count of all common and unique characters in the two strings.

Create a class Main which would get the inputs and call the static method commonChars present in the UserMainCode.

Input and Output Format:

Input consists of two strings.

Output is an integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

a black cow

battle ship

**Sample Output 1:**

2

[Explanation : b, l and a are the common letters between the 2 input strings. But 'a' appears more than once in the 1st string. So 'a' should not be considered while computing the count value.]

**Sample Input 2:**

australia

sri lanka

**Sample Output 2:**

4

**91. Remove 3 Multiples**

Write a program that accepts an ArrayList of integers as input and removes every 3rd element and prints the final ArrayList.

Suppose the given arrayList contains 10 elements remove the 3rd, 6th and 9th elements.

Include a class UserMainCode with a static method ?removeMultiplesOfThree? that accepts an ArrayList as arguement and returns an ArrayList.

Create a class Main which would get the required input and call the static method removeMultiplesOfThree present in the UserMainCode.

Input and Output Format:

The first line of the input consists of an integer n, that corresponds to the number of elements to be added in the ArrayList.

The next n lines consist of integers that correspond to the elements in the ArrayList.

Output consists of an ArrayList of integers.

**Sample Input:**

6

3

1

11

19

17

19

**Sample Output:**

3

1

19

17

**public** **class** UserMainCode {

**public** **static** ArrayList<Integer> findFruitname(ArrayList<Integer> al) {

                        ArrayList<Integer> al2 = **new** ArrayList<Integer>();

                        {

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

**if** ((i + 1) % 3 != 0)

                                                            al2.add(al.get(i));

                                    }

**return** al2;

                        }

            }

}

**92. Anagrams**

Write a program to read two strings and checks if one is an anagram of the other.

An anagram is a word or a phrase that can be created by rearranging the letters of another given word or phrase. We ignore white spaces and letter case. All letters of 'Desperation' can be rearranged to the phrase 'A Rope Ends It'.

Include a class UserMainCode with a static method checkAnagram which accepts the two strings. The return type is boolean which is TRUE / FALSE.

Create a Class Main which would be used to accept the two strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two strings.

Output consists of TRUE / FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

tea

eat

**Sample Output 1:**

TRUE

**Sample Input 2:**

Desperation

A Rope Ends It

**Sample Output 2:**

TRUE

**public** **class** UserMainCode {

**public** **static** **boolean** getAnangram(String s1, String s2)

            {

**boolean** b= **false**;

**char**[] c1=s1.toCharArray();

**char**[] c2=s2.toCharArray();

                        Arrays.*sort*(c1);

                        Arrays.*sort*(c2);

                        String cc=**new** String(c1);

                        String cs=**new** String(c2);

**if**(cc.equals(cs))

                        {

                                    b =  **true**;

                        }

**return** b;

            }

}

**93. String Occurances - II**

Obtain two strings S1,S2 from user as input. Your program should count the number of times S2 appears in S1.

Return the count as output. Note - Consider case.

Include a class UserMainCode with a static method getSubstring which accepts two string variables. The return type is the count.

Create a Class Main which would be used to accept two Input strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two strings with maximum size of 100 characters.

Output consists of an integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

catcowcat

cat

**Sample Output 1:**

2

**Sample Input 2:**

catcowcat

CAT

**Sample Output 2:**

0

**public** **class** UserMainCode {

**public** **static** **int** stringOccurenceII(String s1, String s2) {

**char**[] c1 = s1.toCharArray();

**char**[] c2 = s2.toCharArray();

**int** count = 0, result = 0;

**for** (**int** i = 0; i < s1.length(); i++) {

**for** (**int** j = 0; j < s2.length(); j++) {

**if** (c1[i] == c2[j]) {

                                                            i++;

                                                            count++;

**if** (i > s1.length())

**break**;

                                                }

                                    }

**if** (count == s2.length()) {

                                                result++;

                                                i = i - 1;

                                    }

                                    count = 0;

                        }

**return** result;

            }

}

**94. Shift Left**

Write a program to read a integer array of scores, and return a version of the given array where all the 5's have been removed. The remaining elements should shift left towards the start of the array as needed, and the empty spaces at the end of the array should be filled with 0.

So {1, 5, 5, 2} yields {1, 2, 0, 0}.

Include a class UserMainCode with a static method shiftLeft which accepts the integer array. The return type is modified array.

Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n which is the number of elements followed by n integer values.

Output consists of modified array.

Refer sample output for formatting specifications.

**Sample Input 1:**

7

1

5

2

4

5

3

5

**Sample Output 1:**

1

2

4

3

0

0

0

**public** **class** UserMainCode {

**public** **static** **int**[] shiftLeft(**int** a[]) {

**int** i, k = 0;

**int** n = a.length;

                        ArrayList<Integer> al = **new** ArrayList<Integer>();

**for** (i = 0; i < n; i++) {

**if** (a[i] != 5) {

                                                al.add(a[i]);

                                    }

                        }

**if** (al.size() < n) {

                                    k = n - al.size();

**for** (i = 0; i < k; i++) {

                                                al.add(0);

                                    }

                        }

**int** b[] = **new** **int**[al.size()];

**for** (i = 0; i < al.size(); i++) {

                                    b[i] = al.get(i);

                        }

**return** b;

            }

}

**95. Word Count**

Given a string array (s) with each element in the array containing alphabets or digits. Write a program to add all the digits in every string and return the sum as an integer. If two digits appear simultaneously do not consider it as one number. Ex- For 'Hyderabad 21' consider 2 and 1 as two digits instead of 21 as a number.

Include a class UserMainCode with a static method sumOfDigits which accepts the string array. The return type is the integer formed based on rules.

Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a an integer indicating the number of elements in the string array.

Output consists of a integer .

Refer sample output for formatting specifications.

**Sample Input 1:**

5

AAA1

B2B

4CCC

A5

ABCDE

**Sample Output 1:**

12

**Sample Input 2:**

3

12

C23

5CR2

**Sample Output 2:**

15

**public** **class** UserMainCode {

**public** **static** **int** getSumElements(String[] s1, **int** n) {

**int** sum = 0;

**for** (**int** i = 0; i < s1.length; i++) {

                                    String s = s1[i];

**for** (**int** j = 0; j < s.length(); j++) {

                                                Character ch = s.charAt(j);

**if** (Character.*isDigit*(ch)) {

                                                            sum = sum + Integer.*parseInt*(ch.toString());

                                                }

                                    }

                        }

**return** sum;

            }

}

**96. Programming Logic**

Write a Program that accepts three integer values (a,b,c) and returns their sum. However, if one of the values is 13 then it does not count towards the sum and the next number also does not count. So for example, if b is 13, then both b and c do not count.

Include a class UserMainCode with a static method getLuckySum which accepts three integers. The return type is integer representing the sum.

Create a Class Main which would be used to accept the input integers and call the static method present in UserMainCode.

Input and Output Format:

Input consists of three integers.

Output consists of a single integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

1

2

3

**Sample Output 1:**

6

**Sample Input 2:**

1

2

13

**Sample Output 2:**

3

**Sample Input 3:**

13

3

8

**Sample Output 3:**

8

**public** **class** UserMainCode {

**public** **static** **int** getLuckySum(**int** a, **int** b, **int** c) {

**int** sum = 0;

**if** (a == 13) {

                                    sum = c;

                        } **else** **if** (b == 13) {

                                    sum = a;

                        } **else** **if** (c == 13) {

                                    sum = a + b;

                        } **else** {

                                    sum = a + b + c;

                        }

**return** sum;

            }

}

**97. Prefix Finder**

Given a string array (s) with each element in the array containing 0s and 1s. Write a program to get the number of strings in the array where one String is getting as prefixed in other String in that array .

Example 1: Input: {10,101010,10001,1111} Output =2 (Since 10 is a prefix of 101010 and 10001)

Example 2: Input: {010,1010,01,0111,10,10} Output =3(01 is a prefix of 010 and 0111. Also, 10 is a prefix of 1010) Note: 10 is NOT a prefix for 10.

Include a class UserMainCode with a static method findPrefix which accepts the string array. The return type is the integer formed based on rules.

Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a an integer indicating the number of elements in the string array followed by the array.

Output consists of a integer .

Refer sample output for formatting specifications.

**Sample Input 1:**

4

0

1

11

110

**Sample Output 1:**

3

**public** **class** UserMainCode {

**public** **static** **int** reverseString(String s[]) {

                        LinkedHashSet<String> l1 = **new** LinkedHashSet<String>();

                        ArrayList<String> al = **new** ArrayList<String>();

**int** c = 0;

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

                                    l1.add(s[i]);

                        Iterator<String> it = l1.iterator();

**while** (it.hasNext()) {

                                    al.add(it.next());

                        }

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

                                    String s2 = al.get(i);

**for** (**int** j = 0; j < al.size(); j++) {

                                                String s3 = al.get(j);

**if** ((i != j) && (s3.length() > s2.length())) {

                                                            String s4 = s3.substring(0, s2.length());

**if** (s2.equals(s4))

                                                                        c++;

                                                }

                                    }

                        }

**return** c;

            }

}

**98. Triplets**

Given an integer array, Write a program to find if the array has any triplets. A triplet is a value if it appears 3 consecutive times in the array.

Include a class UserMainCode with a static method checkTripplets which accepts an integer array. The return type is boolean stating whether its a triplet or not.

Create a Class Main which would be used to accept the input arrayand call the static method present in UserMainCode.

Input and Output Format:

Input consists of n+1 integers. The first integer would represent the size of array and the next n integers would have the values.

Output consists of a string stating TRUE or FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

7

3

3

5

5

5

2

3

**Sample Output 1:**

TRUE

**Sample Input 2:**

7

5

3

5

1

5

2

3

**Sample Output 2:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** Triplets(**int**[] a) {

**boolean** b = **false**;

**for** (**int** i = 0; i < a.length - 3; i++) {

**if** ((a[i] == a[i + 1]) && (a[i + 1] == a[i + 2])) {

                                                b = **true**;

                                    }

                        }

**return** b;

            }

}

**99. Commons**

Given two arrays of strings,return the count of strings which is common in both arrays. Duplicate entries are counted only once.

Include a class UserMainCode with a static method countCommonStrings which accepts the string arrays. The return type is the integer formed based on rules.

Create a Class Main which would be used to accept the string arrays and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a an integer indicating the number of elements in the string array followed by the array.

Output consists of a integer .

Refer sample output for formatting specifications.

**Sample Input 1:**

3

a

c

e

3

b

d

e

**Sample Output 1:**

1

**Sample Input 2:**

5

ba

ba

black

sheep

wool

5

ba

ba

have

any

wool

**Sample Output 2:**

2

**public** **class** UserMainCode {

**public** **static** **int** commons(String[] s1, String[] s2) {

**int** count = 0;

                        ArrayList<String> al = **new** ArrayList<String>();

**for** (**int** i = 0; i < s1.length; i++) {

**for** (**int** j = 0; j < s2.length; j++) {

**if** (s1[i].equals(s2[j])) {

**if** (!al.contains(s1[i])) {

                                                                        count++;

                                                                        al.add(s1[i]);

                                                            }

                                                }

                                    }

                        }

**return** count;

            }

}

**100. Initial Format**

Write a program to input a person's name in the format "FirstName LastName" and return the person name in the following format - "LastName, InitialOfFirstName".

Include a class UserMainCode with a static method nameFormatter which accepts a string. The return type (string) should return the expected format.

Create a Class Main which would be used to accept Input String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string that corresponds to a Person's name.

Output consists of a string(person's name in expected format).

Refer sample output for formatting specifications.

**Sample Input :**

Jessica Miller

**Sample Output:**

Miller, J

**public** **class** UserMainCode {

**public** **static** String initialFormat(String s) {

                        StringBuffer sb = **new** StringBuffer();

                        String s1[] = s.split(" ");

                        sb.append(s1[1]).append(",").append(s1[0].charAt(0));

**return** sb.toString();

            }

}

**101. Repeat Front**

Given a string (s) and non negative integer (n) apply the following rules.

    Display the first three characters as front.

    If the length of the string is less than 3, then consider the entire string as front and repeat it n times.

Include a class UserMainCode with a static method repeatFirstThreeCharacters which accepts the string and integer. The return type is the string formed based on rules.

Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string and integer.

Output consists of a string .

Refer sample output for formatting specifications.

**Sample Input 1:**

Coward

2

**Sample Output 1:**

CowCow

**Sample Input 2:**

So

3

**Sample Output 2:**

SoSoSo

**public** **class** UserMainCode {

**public** **static** String repeatFirstThreeChar(String s1, **int** a) {

                        StringBuffer buffer = **new** StringBuffer();

**for** (**int** i = 0; i < a; i++) {

**if** (s1.length() >= 3) {

                                                buffer.append(s1.substring(0, 3));

                                    } **else** {

                                                buffer.append(s1.substring(0, s1.length()));

                                    }

                        }

**return** buffer.toString();

            }

}

**102. Sequence Sum**

Write a program to read a non-negative integer n, and find sum of fibonanci series for n number..

Include a class UserMainCode with a static method getFibonacciSum which accepts the integer value. The return type is integer.

The fibonacci seqence is a famous bit of mathematics, and it happens to have a recursive definition.

The first two values in the sequnce are 0 and 1.

Each subsequent value is the sum of the previous two values, so the whole seqence is 0,1,1,2,3,5 and so on.

You will have to find the sum of the numbers of the Fibonaaci series for a given int n.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a integer.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

**Sample Output 1:**

7

**public** **class** UserMainCode {

**public** **static** **int** sequenceSum(**int** n) {

**int** a = 0, b = 1, c = 0, d = 1;

**for** (**int** i = 3; i <= n; i++) {

                                    c = a + b;

                                    a = b;

                                    b = c;

                                    d = d + c;

                        }

**return** d;

            }

}

**103. E-Mail Validation**

Write a program to read a string and validate the given email-id as input.

Validation Rules:

1. Ensure that there are atleast 5 characters between '@' and '.'

2. There should be only one '.' and one '@' symbol.

3. The '.' should be after the '@' symbol.

4. There must be atleast three characters before '@'.

5. The string after '.' should only be 'com'

Include a class UserMainCode with a static method ValidateEmail which accepts the string. The return type is TRUE / FALSE as per problem.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of TRUE / FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

test@gmail.com

**Sample Output 1:**

TRUE

**Sample Input 2:**

academy@xyz.com

**Sample Output 2:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** emailValidation(String id) {

**boolean** b=**false**;

**if**(id.matches("[a-z]{3,}[@]{1}[a-z]{5}[.]{1}[com]{1}"))

                        {

                                    b=**true**;

                        }

**return** b;

            }

}

**104. Sorted Array**

Write a program to read a string array, remove duplicate elements and sort the array.

Note:

    The check for duplicate elements must be case-sensitive. (AA and aa are NOT duplicates)

    While sorting, words starting with upper case letters takes precedence.

Include a class UserMainCode with a static method orderElements which accepts the string array. The return type is the sorted array.

Create a Class Main which would be used to accept the string arrayand integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n which is the number of elements followed by n string values.

Output consists of the elements of string array.

Refer sample output for formatting specifications.

**Sample Input 1:**

6

AAA

BBB

AAA

AAA

CCC

CCC

**Sample Output 1:**

AAA

BBB

CCC

**Sample Input 2:**

7

AAA

BBB

aaa

AAA

Abc

A

b

**Sample Output 2:**

A

AAA

Abc

BBB

aaa

b

**public** **class** UserMainCode {

**public** **static** String[] sortedArray(String[] s) {

                        HashSet<String> hs = **new** HashSet<String>();

**for** (**int** i = 0; i < s.length; i++) {

                                    hs.add(s[i]);

                        }

                        Iterator<String> it = hs.iterator();

                        s = **new** String[hs.size()];

**int** i = 0;

**while** (it.hasNext()) {

                                    s[i++] = it.next();

                        }

                        Arrays.*sort*(s);

**return** s;

            }

**105. Symmetric Difference**

Write a program to read two integer array and calculate the symmetric difference of the two arrays. Finally Sort the array.

Symmetric difference is the difference of A Union B and A Intersection B ie. [ (A U B) - (A ^ B)]

Union operation merges the two arrays and makes sure that common elements appear only once. Intersection operation includes common elements from both the arrays.

Ex - A={12,24,7,36,14} and B={11,26,7,14}.

A U B ={ 7,11,12,14,24,26,36} and

A ^ B = {7,14}

Symmetric difference of A and B after sorting= [A U B] - [ A ^ B] = {11,12,24,26,36}.

Include a class UserMainCode with a static method getSymmetricDifference which accepts the integer array. The return type is an integer array.

Create a Class Main which would be used to accept the two integer arrays and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n which is the number of elements followed by n integer values. The same sequnce is followed for the next array.

Output consists of sorted symmetric difference array.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

11

5

14

26

3

3

5

3

1

**Sample Output 1:**

1

11

14

26

**public** **class** UserMainCode {

**public** **static** **int**[] getSymmentric(**int**[] a, **int**[] b) {

**int**[] union;

**int**[] inter;

**int**[] result;

**int** count = 0;

**int** max = a.length + b.length;

                        ArrayList<Integer> al = **new** ArrayList<Integer>(max);

                        // union

**for** (**int** i = 0; i < a.length; i++) {

**if** (!al.contains(a[i])) {

                                                ++count;

                                                al.add(a[i]);

                                    }

                        }

**for** (**int** i = 0; i < b.length; i++) {

**if** (!al.contains(b[i])) {

                                                ++count;

                                                al.add(b[i]);

                                    }

                        }

                        union = **new** **int**[count];

**for** (**int** i = 0; i < count; i++) {

                                    union[i] = (**int**) (al.get(i));

                        }

                        Arrays.*sort*(union);

                        // Intersection

                        al = **new** ArrayList<Integer>(max);

                        count = 0;

                        Arrays.*sort*(b);

**for** (**int** i = 0; i < a.length; i++) {

**if** (Arrays.*binarySearch*(b, a[i]) >= 0) {

                                                ++count;

                                                al.add(a[i]);

                                    }

                        }

                        inter = **new** **int**[count];

**for** (**int** i = 0; i < count; i++) {

                                    inter[i] = (**int**) (al.get(i));

                        }

                        Arrays.*sort*(inter);

                        // difference

                        al = **new** ArrayList<Integer>(max);

                        count = 0;

                        Arrays.*sort*(inter);

**for** (**int** i = 0; i < union.length; i++) {

**if** (Arrays.*binarySearch*(inter, union[i]) < 0) {

                                                ++count;

                                                al.add(union[i]);

                                    }

                        }

                        result = **new** **int**[count];

**for** (**int** i = 0; i < count; i++) {

                                    result[i] = (**int**) al.get(i);

                        }

                        Arrays.*sort*(result);

**return** result;

            }

}

**106. Pattern Matcher**

Write a program to read a string and check if it complies to the pattern 'CPT-XXXXXX' where XXXXXX is a 6 digit number. If the pattern is followed, then print TRUE else print FALSE.

Include a class UserMainCode with a static method CheckID which accepts the string. The return type is a boolean value.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output should print TRUE or FALSE .

Refer sample output for formatting specifications.

**Sample Input 1:**

CPT-302020

**Sample Output 1:**

TRUE

**Sample Input 2:**

CPT123412

**Sample Output 2:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** patternValidation(String s1) {

**boolean** b = **false**;

**if** (s1.matches("[CPT-]{4}[0-9]{6}")) {

                                    b = **true**;

                        } **else** {

                                    b = **false**;

                        }

**return** b;

            }

}

**107. Character cleaning**

Write a program to input a String and a character, and remove that character from the given String. Print the final string.

Include a class UserMainCode with a static method removeCharacter which accepts a string and a character. The return type (string) should return the character cleaned string.

Create a Class Main which would be used to accept Input String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string and a character.

Output consists of a string(the character cleaned string).

Refer sample output for formatting specifications.

**Sample Input :**

elephant

e

**Sample Output:**

lphant

**public** **class** UserMainCode {

**public** **static** String characterCleaning(String s1, **char** ch) {

                        StringBuffer sb = **new** StringBuffer(s1);

**for** (**int** i = 0; i < sb.length(); i++) {

**if** (ch == sb.charAt(i)) {

                                                sb.deleteCharAt(i);

                                                i--;

                                    }

                        }

**return** sb.toString();

            }

}

**108. Day of Week**

Write a program to read a string  containing date in DD/MM/YYYY format and prints the day of the week that date falls on.

Return the day in lowercase letter (Ex: monday)

Include a class UserMainCode with a static method getDayOfWeek which accepts the string. The return type is the string.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

02/04/1985

**Sample Output 1:**

tuesday

**public** **class** UserMainCode {

**public** **static** String dayOfWeek(String s1) {

                        SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/yyyy");

                        String day = **null**;

**try** {

                                    Date d = sdf.parse(s1);

                                    SimpleDateFormat sdf1 = **new** SimpleDateFormat("EEEEE");

                                    day = sdf1.format(d).toLowerCase();

                        } **catch** (ParseException e) {

                                    e.printStackTrace();

                        }

**return** day;

            }

}

**109. Add Time**

Write a program to read  two String variables containing time intervals in hh:mm:ss format. Add the two time intervals and return a string in days:hours:minutes:seconds format where DD is number of days.

Hint: Maximum value for hh:mm:ss is 23:59:59

Include a class UserMainCode with a static method addTime which accepts the string values. The return type is the string.

Create a Class Main which would be used to accept the two string values and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two string.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

12:45:30

13:50:45

**Sample Output 1:**

1:2:36:15

**Sample Input 2:**

23:59:59

23:59:59

**Sample Output 2:**

1:23:59:58

**public** **class** UserMainCode {

**public** **static** String setTime(String s1, String s2) {

                        String op = "";

                        SimpleDateFormat sdf = **new** SimpleDateFormat("HH:mm:ss");

                        sdf.setTimeZone(TimeZone.*getTimeZone*("UTC"));

                        sdf.setTimeZone(TimeZone.*getTimeZone*("s1"));

                        sdf.setTimeZone(TimeZone.*getTimeZone*("s2"));

**try** {

                                    Date d1 = sdf.parse(s1);

                                    Date d2 = sdf.parse(s2);

**long** add = d1.getTime() + d2.getTime();

                                    String s = sdf.format(add);

                                    Calendar cal = Calendar.*getInstance*();

                                    cal.setTime(sdf.parse(s));

**int** FindDay = cal.get(Calendar.*DAY\_OF\_MONTH*);

**if** (FindDay > 1)

                                                FindDay = FindDay - 1;

                                    op = FindDay + ":" + s;

                        } **catch** (Exception e) {

                                    e.printStackTrace();

                        }

**return** op;

            }

}

**110. Playing with String - I**

Given a string array and non negative integer (n) apply the following rules.

1. Pick nth character from each String element in the String array and form a new String.

2. If nth character not available in a particular String in the array consider $ as the character.

3. Return the newly formed string.

Include a class UserMainCode with a static method formString which accepts the string and integer. The return type is the string formed based on rules.

Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a an integer which denotes the size of the array followed by the array of strings and an integer (n).

Output consists of a string .

Refer sample output for formatting specifications.

**Sample Input 1:**

4

ABC

XYZ

EFG

MN

3

**Sample Output 1:**

CZG$

**public** **class** UserMainCode {

**public** **static** String get(**int** n, String[] s1, **int** a) {

                        StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < n; i++) {

**if** (s1[i].length() == a) {

                                                String s2 = s1[i];

                                                sb.append(s2.substring(s2.length() - 1));

                                    } **else** {

                                                sb.append('$');

                                    }

                        }

**return** sb.toString();

            }

}

**111. Vowel Check**

Write a program to read a String and check if that String contains all the vowels. Print ?yes? if the string contains all vowels else print ?no?.

Include a class UserMainCode with a static method getVowels which accepts a string. The return type (integer) should return 1 if the String contains all vowels else return -1.

Create a Class Main which would be used to accept Input String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string(?yes? or ?no?).

Refer sample output for formatting specifications.

**Sample Input 1:**

abceiduosp

**Sample Output 1:**

yes

**Sample Input 2:**

bceiduosp

**Sample Output 2:**

no

**public** **class** UserMainCode {

**public** **static** **int** getVowels(String s) {

                        String s2 = s.replaceAll("[^aeiouAEIOU]", "");

                        HashSet<Character> hs = **new** HashSet<Character>();

**for** (**int** i = 0; i < s2.length(); i++) {

                                    hs.add(s2.charAt(i));

                        }

**if** (hs.size() == 5) {

**return** 1;

                        } **else** {

**return** 0;

                        }

            }

}

**112. ISBN Validation**

Write a program to read a string and validate the given ISBN as input.

Validation Rules:

1. An ISBN (International Standard Book Number) is a ten digit code which uniquely identifies a book.

2. To verify an ISBN you calculate 10 times the first digit, plus 9 times the second digit, plus 8 times the third ..all the way until you add 1 times the last digit.

If the final number leaves no remainder when divided by 11 the code is a valid ISBN.

Example 1:

Input:0201103311

Calculation: 10\*0 + 9\*2 + 8\*0 + 7\*1 + 6\*1 + 5\*0 + 4\*3 + 3\*3 + 2\*1 + 1\*1 = 55.

55 mod 11 = 0

Hence the input is a valid ISBN number

Output: true

Include a class UserMainCode with a static method validateISBN which accepts the string. The return type is TRUE / FALSE as per problem.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of TRUE / FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

0201103311

**Sample Output 1:**

TRUE

**public** **class** UserMainCode {

**public** **static** **boolean** ISBNValidation(String ip) {

**boolean** b = **false**;

**int** sum = 0;

**for** (**int** i = 0, j = ip.length(); i < ip.length(); i++, j--) {

                                    String s = String.*valueOf*(ip.charAt(i));

**int** n = Integer.*parseInt*(s);

                                    sum = sum + (n \* j);

                        }

**if** (sum % 11 == 0) {

                                    b = **true**;

                        }

**return** b;

            }

}

**113. Regular Expression – 1(Validate)**

Given a string (s) apply the following rules.

1. String should be only four characters long.

2. First character can be an alphabet or digit.

3. Second character must be uppercase 'R'.

4. Third character must be a number between 0-9.

If all the conditions are satisifed then print TRUE else print FALSE.

Include a class UserMainCode with a static method validate which accepts the string. The return type is the boolean formed based on rules.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of TRUE or FALSE .

Refer sample output for formatting specifications.

**Sample Input 1:**

vR4u

**Sample Output 1:**

TRUE

**Sample Input 2:**

vRau

**Sample Output 2:**

FALSE

**Sample Input 3:**

vrau

**Sample Output 3:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** validate(String s1) {

**if** (s1.matches("[a-zA-Z0-9]{1}[R]{1}[0-9]{1}[a-zA-Z]{1}")) {

**return** **true**;

                        } **else** {

**return** **false**;

                        }

            }

}

**114. Date Format**

Write a program to read  two String variables in DD-MM-YYYY.Compare the two dates and return the older date in 'MM/DD/YYYY' format.

Include a class UserMainCode with a static method findOldDate which accepts the string values. The return type is the string.

Create a Class Main which would be used to accept the two string values and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two string.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

05-12-1987

8-11-2010

**Sample Output 1:**

12/05/1987

**public** **class** UserMainCode {

**public** **static** String dateFormat(String s1, String s2) {

                        String s3 = "";

                        String s4 = "";

**long** y = 0;

**long** y1 = 0;

                        SimpleDateFormat sdf1 = **new** SimpleDateFormat("dd-MM-yyyy");

                        SimpleDateFormat sdf2 = **new** SimpleDateFormat("MM/dd/yyyy");

**try** {

                                    Date d1 = sdf1.parse(s1);

                                    Date d2 = sdf1.parse(s2);

                                    Calendar cal = Calendar.*getInstance*();

                                    cal.setTime(d1);

                                    y = cal.getTimeInMillis();

                                    cal.setTime(d2);

                                    y1 = cal.getTimeInMillis();

                                    s3 = sdf2.format(d1);

                                    s4 = sdf2.format(d2);

                        } **catch** (ParseException e) {

                                    e.printStackTrace();

                        }

**if** (y < y1)

**return** s3;

**else**

**return** s4;

            }

}

**115. Swap Characters**

Write a program to input a String and swap the every 2 characters in the string. If size is an odd number then keep the last letter as it is. Print the final swapped string.

Include a class UserMainCode with a static method swapCharacter which accepts a string. The return type (String) should return the character swapped string.

Create a Class Main which would be used to accept Input String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

TRAINER

**Sample Output 1:**

RTIAENR

**Sample Input 2:**

TOM ANDJERRY

**Sample output 2:**

OT MNAJDREYR

**public** **class** UserMainCode {

**public** **static** String swapCharater(String s1) {

                        StringBuffer sb = **new** StringBuffer();

**int** l = s1.length();

**if** (l % 2 == 0) {

**for** (**int** i = 0; i < s1.length() - 1; i = i + 2) {

**char** a = s1.charAt(i);

**char** b = s1.charAt(i + 1);

                                                sb.append(b).append(a);

                                    }

**return** sb.toString();

                        } **else** {

**for** (**int** i = 0; i < s1.length() - 1; i = i + 2) {

**char** a = s1.charAt(i);

**char** b = s1.charAt(i + 1);

                                                sb.append(b).append(a);

                                    }

                                    sb.append(s1.charAt(l - 1));

                        }

**return** sb.toString();

            }

}

**116. Regular Expression ? 2 (Age Validator)**

Given the age of a person as string, validate the age based on the following rules.

1. Value should contain only numbers.

2. Value should be non-negative.

3. Value should be in the range of 21 to 45'.

If all the conditions are satisifed then print TRUE else print FALSE.

Include a class UserMainCode with a static method ValidateAge which accepts the string. The return type is the boolean formed based on rules.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of TRUE or FALSE .

Refer sample output for formatting specifications.

**Sample Input 1:**

23

**Sample Output 1:**

TRUE

**Sample Input 2:**

-34

**Sample Output 2:**

FALSE

**Sample Input 3:**

3a

**Sample Output 3:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** ageValidator(String s) {

**if**(s.matches("[2]{1}[0-9]{1}|[3]{1}[0-9]{1}|[4]{1}[0-5]{1}"))

                        {

**return** **true**;

                        }

**else**

                        {

**return** **false**;

                        }

            }

}

**117. Interest Calculation**

Write a  program to calculate amount of the acccount holders based on the below mentioned prototype:

1. Read account details from the User. The details would include id, DOB (date of birth) and amount in the given order. The datatype for id is string, DOB is string and amount is integer.

2. You decide to build two hashmaps. The first hashmap contains employee id as key and DOB as value, and the second hashmap contains same employee ids as key and amount as value.

3. Rate of interest as on 01/01/2015:

    a. If the age greater than or equal to 60 then interest rate is 10% of Amount.

    b.If the age less then to 60 and greater than or equal to 30 then interest rate is 7% of Amount.

    v. If the age less then to 30 interest rate is 4% of Amount.

4. Revised Amount= principle Amount + interest rate.

5.  You decide to write a function calculateInterestRate which takes the above hashmaps as input and returns the treemap  as output. Include this function in class UserMainCode.

Create a Class Main which would be used to read employee details in step 1 and build the two hashmaps. Call the static method present in UserMainCode.

Input and Output Format:

Input consists of account details. The first number indicates the size of the acoount. The next three values indicate the user id, DOB and amount. The Employee DOB format is ?dd-mm-yyyy?

Output consists of the user id and the amount for each user one in a line.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

SBI-1010

20-01-1987

10000

SBI-1011

03-08-1980

15000

SBI-1012

05-11-1975

20000

SBI-1013

02-12-1950

30000

**Sample Output 1:**

SBI-1010:10400

SBI-1011:16050

SBI-1012:21400

SBI-1013:33000

**118. Discount Rate Calculation**

Write a  program to calculate discount of the acccount holders based on the transaction amount and registration date using below mentioned prototype:

1. Read account details from the User. The details would include id, DOR (date of registration) and transaction amount in the given order. The datatype for id is string, DOR is string and transaction amount is integer.

2. You decide to build two hashmaps. The first hashmap contains employee id as key and DOR as value, and the second hashmap contains same employee ids as key and amount as value.

3. Discount Amount as on 01/01/2015:

    a. If the transaction amount greater than or equal to 20000 and registration greater than or equal to 5     year then discount rate is 20% of transaction amount.

    b. If the transaction amount greater than or equal to 20000 and registration less then to 5 year then     discount rate is 10% of transaction amount.

    c. If the transaction amount less than to 20000 and registration greater than or equal to 5 year then     discount rate is 15% of transaction amount.

    d. If the transaction amount less than to 20000 and registration less then to 5 year then discount rate     is 5% of transaction amount.

4. You decide to write a function calculateDiscount which takes the above hashmaps as input and returns the treemap  as output. Include this function in class UserMainCode.

Create a Class Main which would be used to read employee details in step 1 and build the two hashmaps. Call the static method present in UserMainCode.

Input and Output Format:

Input consists of transaction details. The first number indicates the size of the employees. The next three values indicate the user id, user DOR and transaction amount. The DOR (Date of Registration) format is ?dd-mm-yyyy?

Output consists of a string which has the user id and discount amount one in a line for each user.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

A-1010

20-11-2007

25000

B-1011

04-12-2010

30000

C-1012

11-11-2005

15000

D-1013

02-12-2012

10000

**Sample Output 1:**

A-1010:5000

B-1011:3000

C-1012:2250

D-1013:500

**119. Average of Elements in Hashmap**

Given a method with a HashMap as input. Write code to find out avg of all values whose keys are even numbers. Round the average to two decimal places and return as output.

[Hint : If the average is 5.901, the rounded average value is 5.9 . It the average is 6.333, the rounded average value is 6.33 . ]

Include a class UserMainCode with a static method avgOfEven which accepts a HashMap as input.

The return type of the output is a floating point value which is the average of all values whose key elements are even numbers.

Create a class Main which would get the input and call the static method avgOfEven present in the UserMainCode.

Input and Output Format:

Input consists of the number of elements in the HashMap and the HashMap.

Output is a floating point value that corresponds to the average.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

1

2.3

2

4.1

6

6.2

**Sample Output 1:**

5.15

**Sample Input 2:**

3

9

3.1

4

6.3

1

2.6

**Sample Output 2:**

6.3

**public** **class** UserMainCode {

**public** **static** String avgElementHashMap(HashMap<Integer, Float> hm) {

**float** sum = 0;

**int** count = 0;

                        DecimalFormat df = **new** DecimalFormat("#.00");

                        Iterator<Integer> it = hm.keySet().iterator();

**while** (it.hasNext()) {

**int** y = (it.next());

**if** (y % 2 == 0) {

                                                sum = (**float**) (sum + hm.get(y));

                                                count++;

                                    }

                        }

**float** d = sum / count;

**return** df.format(d);

            }

}

**120. Calculate Average ? Hash Map**

Write amethod that accepts the input data as a hash map and finds out the avg of all values whose keys are odd numbers.

Include a class UserMainCode with a static method calculateAverage which accepts a HashMap and the size of the HashMap. The return type (Double) should return the calculated average. Round the average to two decimal places and return it.

Create a Class Main which would be used to accept Input values and store it as a hash map, and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n corresponds to number of hash map values, followed by 2n values. (index followed by value).

Output consists of a Double.

Refer sample input and output for formatting specifications.

**Sample Input :**

4

1

3.41

2

4.1

3

1.61

4

2.5

**Sample Output :**

2.51

**public** **class** UserMainCode {

**public** **static** String avgElementHashMap(HashMap<Integer, Float> hm) {

**float** sum = 0;

**int** count = 0;

                        DecimalFormat df = **new** DecimalFormat("#.00");

                        Iterator<Integer> it = hm.keySet().iterator();

**while** (it.hasNext()) {

**int** y = (it.next());

**if** (y % 2 != 0) {

                                                sum = (**float**) (sum + hm.get(y));

                                                count++;

                                    }

                        }

**float** d = sum / count;

**return** df.format(d);

            }

}

**121. String Splitter**

Write a program which would accept a string and a character as a delimiter. Apply the below rules

1. Using the delimiter, split the string and store these elements in array.

2. Reverse each element of the string and convert it into lowercase.

Include a class UserMainCode with a static method manipulateLiteral which accepts the string and character. The return type is the string array formed.

Create a Class Main which would be used to accept the string and characterand call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string and character.

Output consists of a string array.

Refer sample output for formatting specifications.

**Sample Input 1:**

AAA/bba/ccc/DDD

/

**Sample Output 1:**

aaa

abb

ccc

ddd

**public** **class** UserMainCode {

**public** **static** String[] stringSplitter(String s1, **char** c) {

            String[] op=s1.split(String.*valueOf*(c));

**for** (**int** i = 0; i < op.length; i++) {

                        op[i]=op[i].toLowerCase();

            }

**return** op;

            }

}

**122. Count Sequential Characters**

Get a string as input and write code to count the number of characters which gets repeated 3 times consecutively and return that count (ignore case). If no character gets repeated 3 times consecutively return -1.

Include a class UserMainCode with a static method countSequentialChars which accepts a string as input.

The return type of the output is the repeat count.

Create a class Main which would get the input and call the static method countSequentialChars present in the UserMainCode.

Input and Output Format:

Input consists a string.

Output is an integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

abcXXXabc

**Sample Output 1:**

1

**Sample Input 2:**

aaaxxyzAAAx

**Sample Output 2:**

2

**public** **class** UserMainCode {

**public** **static** **int** countSquential(String s1) {

**int** c = 0, n = 0;

**for** (**int** i = 0; i < s1.length() - 1; i++) {

**if** (s1.charAt(i) == s1.charAt(i + 1))

                                                n++;

**else**

                                                n = 0;

**if** (n == 2)

                                                c++;

                        }

**return** c;

            }

}

**123. Vowel Count**

Write a program to read a string and count the number of vowels present in it.

Include a class UserMainCode with a static method tellVowelCount which accepts the string. The return type is the integer giving out the count of vowels.

Note: The check is case-insensitive.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

NewYork

**Sample Output 1:**

2

**Sample Input 2:**

Elephant

**Sample Output 2:**

3

**public** **class** UserMainCode {

**public** **static** **int** countVowels(String s1) {

**int** count=0;

                        s1= s1.replaceAll("[^aeiouAEIOU]", "");

                        count=s1.length();

**return** count;

            }

}

**124. Length of the Largest Chunk**

Write a program to read a string and find the length of the largest chunk in the string. If there are no chunk print ?No chunks? else print the length.

NOTE: chunk is the letter which is repeating 2 or more than 2 times.

Include a class UserMainCode with a static method largestChunk which accepts a string. The return type (Integer) should return the length of the largest chunk if the chunk is present, else return -1.

Create a Class Main which would be used to accept Input String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

You are toooo good

**Sample Output 1:**

4

(Because the largest chunk is letter 'o' which is repeating 4 times)

**Sample Input 2:**

who are u

**Sample Output 2:**

No chunks

**public** **class** UserMainCode {

**public** **static** **int** getlargestChunk(String s1) {

                        StringTokenizer st = **new** StringTokenizer(s1, "");

**int** max = 0;

**while** (st.hasMoreTokens()) {

                                    String s = st.nextToken();

                                    StringBuffer sb = **new** StringBuffer(s);

**for** (**int** i = 0; i < sb.length(); i++) {

**int** count = 0;

**for** (**int** j = i + 1; j < sb.length(); j++) {

**if** (sb.charAt(i) == sb.charAt(j)) {

                                                                        count++;

                                                            } **else** {

**break**;

                                                            }

                                                }

**if** (count > max) {

                                                            max = count + 1;

                                                }

                                    }

                        }

**if** (max == 0) {

**return** -1;

                        } **else** {

**return** max;

                        }

            }

}

**125. Playing with String - II**

Write a program to accept a string array as input, convert all the elements into lowercase and sort the string array. Display the sorted array.

Include a class UserMainCode with a static method sortArray which accepts the string array. The return type is the string array formed based on requirement.

Create a Class Main which would be used to accept the string array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a an integer which denotes the size of the array followed by the array of strings,

Output consists of a string array.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

AAA

BB

CCCC

A

ABCDE

**Sample Output 1:**

a

aaa

abcde

bb

cccc

**public** **class** UserMainCode

{

**public** **static** String[] sortArray(String s[])

            {

**int**  i=0;

                        ArrayList<String> al = **new** ArrayList<String>();

**for** (i = 0; i < s.length; i++)

                        {

                                    al.add(s[i]);

                        }

                        String s1[] = **new** String[al.size()];

                        Collections.*sort*(al);

**for** (i = 0; i < al.size(); i++)

                        {

                                    s1[i]=al.get(i).toLowerCase();

                        }

**return** s1;

            }

}

**126. Find Digits**

For a given double number with atleast one decimal value, Write a program to compute the number of digits before and after the decimal point in the following format ? noOfDigitsBeforeDecimal:noOfDigitsAfterDecimal.

Note: Ignore zeroes at the end of the decimal (Except if zero is the only digit after decimal. Refer Example 2 and 3)

Include a class UserMainCode with a static method findNoDigits which accepts the decimal value. The return type is string.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a double.

Output consists of string.

Refer sample output for formatting specifications.

**Sample Input 1:**

843.21

**Sample Output 1:**

3:2

**Sample Input 2:**

20.130

**Sample Output 2:**

2:2

**Sample Input 3:**

20.130

**Sample Output 3:**

2:2

**public** **class** UserMainCode {

**public** **static** String findNoDigits(**double** a) {

                        String b = String.*valueOf*(a);

                        StringBuffer sb = **new** StringBuffer();

                        String[] st=b.split("\\.");

                        String c = st[0];

                        String d = st[1];

**int** x = c.length();

**int** y = d.length();

                        sb.append(x).append(':').append(y);

**return** sb.toString();

            }

**127. Unique Characters in a string**

Write a program that takes a string and returns the number of unique characters in the string. If the given string doest not contain any unique characters return -1

Include a class UserMainCode with a static method uniqueCounter which accepts a string as input.

The return type of the output is the count of all unique characters in the strings.

Create a class Main which would get the input and call the static method uniqueCounter present in the UserMainCode.

Input and Output Format:

Input consists a string.

Output is an integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

HelloWorld

**Sample Output 1:**

5

**Sample Input 2:**

coco

**Sample Output 2:**

-1

**public** **class** UserMainCode {

**public** **static** **int** uniqueCounter(String s1) {

                        StringBuffer sb = **new** StringBuffer(s1);

**for** (**int** i = 0; i < sb.length(); i++) {

**int** count = 0;

**for** (**int** j = i + 1; j < sb.length(); j++) {

**if** (sb.charAt(i) == sb.charAt(j)) {

                                                            sb.deleteCharAt(j);

                                                            j--;

                                                            count++;

                                                }

                                    }

**if** (count >= 1) {

                                                sb.deleteCharAt(i);

                                                i--;

                                    }

                        }

**if** (sb.length() == 0) {

**return** -1;

                        } **else**

**return** sb.length();

            }

}

**128. Median Calculation**

Write a program to accept an int array as input, and calculate the median of the same.

Median Calculation Procedure:

    Sort the sequence of numbers.

    The total number count is odd, Median will be the middle number.

    The total number count is even, Median will be the average of two middle numbers, After calculating the average, round the number to nearest integer.

Include a class UserMainCode with a static method calculateMedian which accepts the int array. The return type is the integer which would be the median.

Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a an integer which denotes the size of the array followed by the array of integers.

Output consists of a integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

7

1

2

1

4

7

1

2

**Sample Output 1:**

2

**Sample Input 2:**

6

52

51

81

84

60

88

**Sample Output 2:**

71

**public** **class** UserMainCode {

**public** **static** **float** calculateMedian(**int** a[]) {

**float** mid;

**int** s = a.length;

                        Arrays.*sort*(a);

**if** (s % 2 != 0) {

                                    mid = a[(s - 1) / 2];

                        } **else** {

                                    mid = (a[s / 2] + a[(s / 2) - 1]) / 2;

                        }

**return** mid;

            }

}

**129. Employees & Designations**

A Company wants to obtain employees of a particular designation. You have been assigned as the programmer to build this package. You would like to showcase your skills by creating a quick prototype. The prototype consists of the following steps:

    Read Employee details from the User. The details would include name and designaton in the given order. The datatype for name and designation is string.

    Build a hashmap which contains the name as key and designation as value.

    You decide to write a function obtainDesignation which takes the hashmap and designation as input and returns a string List of employee names who belong to that designation as output. Include this function in class UserMainCode. Display employee name's in ascending order.

Create a Class Main which would be used to read employee details in step 1 and build the hashmap. Call the static method present in UserMainCode.

Input and Output Format:

Input consists of employee details. The first number indicates the size of the employees. The next two values indicate the employee name employee designation. The last string would be the designation to be searched.

Output consists of a array values containing employee names.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

Manish

MGR

Babu

CLK

Rohit

MGR

Viru

PGR

MGR

**Sample Output 1:**

Manish

Rohit

**public** **class** UserMainCode {

**public** **static** String[] obtainDesignation(HashMap<String, String> hm, **int** n,

                                    String s) {

**int** k = 0, i = 0;

                        String s1[] = **new** String[n];

**for** (Map.Entry e : hm.entrySet()) {

**if** (e.getValue().equals(s)) {

                                                s1[k] = (String) e.getKey();

                                                k++;

                                    }

                        }

**for** (i = 0; i < s1.length; i++)

**return** s1;

**return** s1;

            }

}

**130. Grade Calculator**

A School wants to give assign grades to its students based on their marks. You have been assigned as the programmer to automate this process. You would like to showcase your skills by creating a quick prototype. The prototype consists of the following steps:

    Read student details from the User. The details would include name, mark in the given order. The datatype for name is string, mark is float.

    You decide to build a hashmap. The hashmap contains name as key and mark as value.

**BUSINESS RULE:**

1. If Mark is less than 60, then grade is FAIL.

2. If Mark is greater than or equal to 60, then grade is PASS.

Note: FAIL/PASS should be in uppercase.

Store the result in a new Hashmap with name as Key and grade as value.

4. You decide to write a function calculateGrade which takes the above hashmap as input and returns the hashmap as output. Include this function in class UserMainCode.

Create a Class Main which would be used to read student details in step 1 and build the hashmap. Call the static method present in UserMainCode.

Input and Output Format:

Input consists of student details. The first number indicates the size of the students. The next two values indicate the name, mark.

Output consists of a name and corresponding grade for each student.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

Avi

76.36

Sunil

68.42

Raja

36.25

**Sample Output 1:**

Avi

PASS

Sunil

PASS

Raja

FAIL

**public** **class** UserMainCode {

**public** **static** HashMap<String, String> examResult(Map<String, Double> m1) {

                        HashMap<String, String> m2 = **new** HashMap<String, String>();

                        String s1 = **new** String();

                        Double n = 0.0;

                        Iterator<String> i = m1.keySet().iterator();

**while** (i.hasNext()) {

                                    s1 = (String) i.next();

                                    n = m1.get(s1);

**if** (n >= 60) {

                                                m2.put(s1, "PASS");

                                    } **else**

                                                m2.put(s1, "FAIL");

                        }

**return** m2;

            }

}

**131. Sequence in Array**

Write a program to accept an int array as input, and check if [1,2,3] appears somewhere in the same sequence.

Include a class UserMainCode with a static method searchSequence which accepts the int array. The return type is a boolean which returns true or false.

Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a an integer which denotes the size of the array followed by the array of integers.

Output should print true or false.

Refer sample output for formatting specifications.

**Sample Input 1:**

9

11

-2

5

1

2

3

4

5

6

**Sample Output 1:**

TRUE

**Sample Input 2:**

6

-2

5

1

3

2

6

**Sample Output 2:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** sequenceInArray(**int**[] a) {

**boolean** b = **false**;

**for** (**int** i = 0; i < a.length - 3; i++) {

**if** (a[i] == 1 && a[i + 1] == 2 && a[i + 2] == 3)

                                                b = **true**;

                        }

**return** b;

            }

}

**132. DOB - Validation**

Write a program to validate the Date of Birth given as input in String format (MM/dd/yyyy) as per the validation rules given below. Return true for valid dates else return false.

1. Value should not be null

2. month should be between 1-12, date should be between 1-31 and year should be a four digit number.

Include a class UserMainCode with a static method ValidateDOB which accepts the string. The return type is TRUE / FALSE.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of TRUE / FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

12/23/1985

**Sample Output 1:**

TRUE

**Sample Input 2:**

31/12/1985

**Sample Output 2:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** ValidateDOB(String s1) {

**if** (s1.matches("([1-12]{2})/([1-31]{2})/([0-9]{4})")) {

**return** **true**;

                        } **else** {

**return** **false**;

                        }

            }

}

**133. Asterisk & Characters**

Write a program to read a string and return true or false based on the below rule:

1. Return true if for every '\*' in the string, there are same characters both side immediately before and after the star, else return false.

Include a class UserMainCode with a static method scanStarNeighbors which accepts the string. The return type is the boolean TRUE or FALSE based on the rule.

Note: The check is case-insensitive.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of TRUE or FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

Hello\*World

**Sample Output 1:**

FALSE

**Sample Input 2:**

Welcome\*elizabeth

**Sample Output 2:**

TRUE

**public** **class** UserMainCode {

**public** **static** **boolean** astrikCharacter(String s1) {

**boolean** b = **false**;

                        String s[] = s1.split("\\\*");

**for** (**int** i = 0; i < s.length; i++) {

                                    String s3 = s[0].substring(s[0].length() - 1);

                                    String s4 = s[1].substring(0, 1);

**if** (s3.equalsIgnoreCase(s4))

                                                b = **true**;

                        }

**return** b;

            }

}

**134. Experience Validator**

Write a program to valiadate the experience of an employee.

    An employee who has recently joined the organization provides his year of passing and total number of years of experience in String format. Write code to validate his experience against the current date.

1) Input consists of two String first represent the year of passed out and the second string reperesent the year of experience.

2) create a function with  name validateExp which accepts two string as input and boolean as output.

3) The difference between current year and year of pass should be more than or equal to Experience

Return true if all condition are true.

Note:Conside 2015 as the current year.

Include a class UserMainCode with the static function validateExp

Create a Class Main which would be used to accept the boolean and call the static method present in UserMainCode.

Input and Output Formate:

Input consists of two Strings.

output will display true if the given data are correct.

**Sample Input:**

2001

5

**Sample Output:**

TRUE

**public** **class** UserMainCode {

**public** **static** **boolean** expValidator(String s, String s1) {

**int** y1 = Integer.*parseInt*(s);

                        Date d = **new** Date();

                        Calendar cal = Calendar.*getInstance*();

**int** y2 = cal.get(Calendar.*YEAR*);

**int** y = Math.*abs*(y1-y2);

**int** e = Integer.*parseInt*(s1);

**if**(y>=e)

**return** **true**;

**else**

**return** **false**;

            }

}

**135. Occurance Count**

Write a program to read a string that contains a sentence and read a word. Check the number of occurances of that word in the sentence.

Include a class UserMainCode with a static method countWords which accepts the two strings. The return type is the integer giving the count.

Note: The check is case-sensitive.

Create a Class Main which would be used to accept the two strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two strings.

Output consists of count indicating the number of occurances.

Refer sample output for formatting specifications.

**Sample Input 1:**

Hello world Java is best programming language in the world

world

**Sample Output 1:**

2

**Sample Input 2:**

hello world

World

**Sample Output 2:**

0

**public** **class** UserMainCode {

**public** **static** **int** countWords(String s1, String s2) {

**int** count = 0;

                        String s[] = s1.split(" ");

**for** (**int** i = 0; i < s.length; i++) {

**if** (s[i].equals(s2)) {

                                                count++;

                                    }

                        }

**return** count;

            }

}

**136. ArrayList to String Array**

Write a program that performs the following actions:

    Read n strings as input.

    Create an arraylist to store the above n strings in this arraylist.

    Write a function convertToStringArray which accepts the arraylist as input.

    The function should sort the elements (strings) present in the arraylist and convert them into a string array.

    Return the array.

Include a class UserMainCode with the static method convertToStringArray which accepts an arraylist and returns an array.

Create a Class Main which would be used to read n strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of n+1 integers. The first integer denotes the size of the arraylist, the next n strings are values to the arraylist.

Output consists of an arrayas per step 4.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

a

d

c

b

**Sample Output 1:**

a

b

c

d

**public** **class** Main {

**public** **static** **void** main(String[] args) {

                        Scanner sc = **new** Scanner(System.*in*);

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

**int** n = sc.nextInt();

                        sc.nextLine();

**for** (**int** i = 0; i < n; i++) {

                                    String s = sc.nextLine();

                                    list.add(s);

                        }

                        String[] str = UserMainCode.*convertToString*(list);

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

                                    System.*out*.println(str[i]);

            }

}

**public** **class** UserMainCode {

**public** **static** String[] convertToString(ArrayList<String> list) {

                        java.util.Collections.*sort*(list);

                        String s[] = **new** String[list.size()];

**return** list.toArray(s);

            }

}

**137. Name Shrinking**

Write a program that accepts a string as input and converts the first two names into dot-separated initials and printa the output.

Input string format is 'fn mn ln'.   Output string format is 'ln [mn's 1st character].[fn's 1st character]'

Include a class UserMainCode with a static method getFormatedString which accepts a string. The return type (String) should return the shrinked name.

Create a Class Main which would be used to accept Input String and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a String.

Refer sample output for formatting specifications.

**Sample Input:**

Sachin Ramesh Tendulkar

**Sample Output:**

Tendulkar R.S

**public** **class** UserMainCode {

**public** **static** String nameShrinking(String s1) {

                        StringBuffer sb = **new** StringBuffer();

                        String s[] = s1.split(" ");

                        sb.append(s[s.length - 1]).append(" ").append(s[1].charAt(0))

                                                .append(".").append(s[0].charAt(0));

**return** sb.toString();

            }

}

**138. Regular Expressions – III(CatCow)**

Write a program to read two strings S1 & S2, compute the number of times that S2 appears in S1.

Include a class UserMainCode with a static method searchString which accepts the two strings. The return type is the integer giving the count.

Note: The check is case-insensitive.

Create a Class Main which would be used to accept the two strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two strings.

Output consists of count indicating the number of occurances.

Refer sample output for formatting specifications.

**Sample Input 1:**

Catcowcat

cat

**Sample Output 1:**

2

**Sample Input 2:**

Catcowcat

catp

**Sample Output 2:**

0

**public** **class** UserMainCode {

**public** **static** **int** regularExpIII(String s1, String s2) {

**int** count = 0;

**if** (s1.contains(s2)) {

                                    count++;

                        }

**return** count;

            }

}

**139. State ID generator**

Write a program to generate the state ID.

     1)Read n Strings as input(as State Name).

     2)Create a String Array to Store the above Input.

     3)Write a function getStateId which accepts String Array as input.

     4)Create a HashMap which stores state name as key and state Id as Value.

     5)The function getStateId returns the HashMap to the Main Class.

Include UserMainCode Class With static method getStateId which accepts String array and return a hashmap.

Create a Class Main which would be used to read n strings and call the static method present in UserMainCode.

**Input and Output Format:**

Input Consists of an integer n denotes the size of the string array.

Output consists of an HashMap displayed in the string array order.

**Sample Input 1:**

3

Kerala

Gujarat

Goa

**Sample Output 1:**

KER:Kerala

GUJ:Gujarat

GOA:Goa

**public** **class** UserMainCode {

**public** **static** HashMap<String, String> getID(String[] s1) {

                        LinkedHashMap<String, String> hm = **new** LinkedHashMap<String, String>();

**for** (**int** i = 0; i < s1.length; i++) {

                                    hm.put(s1[i].substring(0, 3).toUpperCase(), s1[i]);

                        }

**return** hm;

            }

}

**140. Odd Digit Sum**

Write a program to input a String array. The input may contain digits and alphabets (?de5g4G7R?). Extract odd digits from each string and find the sum and print the output.

For example, if the string is "AKj375A" then take 3+7+5=15 and not as 375 as digit.

Include a class UserMainCode with a static method oddDigitSum which accepts a string array and the size of the array. The return type (Integer) should return the sum.

Create a Class Main which would be used to accept Input Strings and call the static method present in UserMainCode.

Assume maximum length of array is 20.

Input and Output Format:

Input consists of an integer n, corresponds to the number of strings, followed by n Strings.

Output consists of an Integer.

Refer sample output for formatting specifications.

**Sample Input :**

3

cog2nizant1

al33k

d2t4H3r5

**Sample Output :**

15

(1+3+3+3+5)

**public** **class** UserMainCode {

**public** **static** Integer oddDigitSum(String s1[], **int** size) {

**int** sum = 0;

                        String s2[] = **new** String[size];

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

                                    s2[i] = s1[i].replaceAll("[a-zA-Z]", "");

                        }

**int** s3[] = **new** **int**[size];

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

                                    s3[i] = Integer.*parseInt*(s2[i]);

                        }

                        ArrayList<Integer> al = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < s3.length; i++) {

                                    al.add(s3[i]);

                        }

                        Iterator<Integer> itr = al.iterator();

                        {

**while** (itr.hasNext()) {

**int** x = itr.next();

**int** rem;

**while** (x != 0) {

                                                            rem = x % 10;

**if** (rem % 2 != 0) {

                                                                        sum = sum + rem;

                                                            }

                                                            x = x / 10;

                                                }

                                    }

**return** sum;

                        }

            }

}

**141. Strings Processing**

Write a program to read a string that contains comma separated fruit names and also a number N. Pick the nth fruit and return it. If the total number of elements are less than the number specified in N, then return the last element.

Include a class UserMainCode with a static method findFruitName which accepts the the string and the number n. The return type is the string which has the fruit name.

Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string and integer.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Apple,Banana,Orange

2

**Sample Output 1:**

Banana

**Sample Input 2:**

Apple,Banana,Orange

4

**Sample Output 2:**

Orange

**public** **class** UserMainCode {

**public** **static** String Fruit(String s1, **int** n) {

                        String s[] = s1.split(",");

                        StringBuffer sb = **new** StringBuffer();

**if** (n > s.length)

                                    sb.append(s[s.length - 1]);

**else** {

**for** (**int** i = 0; i < s.length; i++) {

**if** (i + 1 == n) {

                                                            sb.append(s[i]);

                                                }

                                    }

                        }

**return** sb.toString();

            }

}

**142. ArrayList to String Array**

Write a program that performs the following actions:

1.Read m strings as input (fruit names).

2.Create an arraylist to store the above m strings in this arraylist.

3.Read n strings as input (fruit names).

4.Create an arraylist to store the above n strings in this arraylist.

5.Write a function fruitSelector which accepts the arraylists as input.

6.Remove all fruits whose name ends with 'a' or 'e' from first arrayList and remove all fruits whose name begins  with 'm' or 'a' from second arrayList then combine the two lists and return the final output as a String array.

7.If the array is empty the program will print as ?No fruit found?

Include a class UserMainCode with the static method fruitSelector which accepts the two arraylists and returns an array.

Create a Class Main which would be used to read n strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer (m) denoting the size of first arraylist. The next m elements would be the values of the first arraylist. The next input would be n denoting the size of the second arraylist. The next n elements would be the values of the second arraylist.

Output consists of an array as per step 6. Refer sample output for formatting specifications.

**Sample Input 1:**

3

Apple

Cherry

Grapes

4

Orange

Mango

Melon

Apple

**Sample Output 1:**

Cherry

Grapes

Orange

**public** **class** UserMainCode {

**public** **static** String[] fruitSelector(ArrayList<String> a,

                                    ArrayList<String> b) {

                        ArrayList<String> op = **new** ArrayList<String>();

                        String k = "";

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

                                    k = a.get(i);

**int** len = k.length() - 1;

**if** (String.*valueOf*(k.charAt(len)).equalsIgnoreCase("a")

                                                            || String.*valueOf*(k.charAt(len)).equalsIgnoreCase("e")) {

**continue**;

                                    } **else** {

                                                op.add(k);

                                    }

                        }

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

                                    k = b.get(i);

**if** (String.*valueOf*(k.charAt(0)).equalsIgnoreCase("m")

                                                            || String.*valueOf*(k.charAt(0)).equalsIgnoreCase("a")) {

**continue**;

                                    } **else** {

                                                op.add(k);

                                    }

                        }

**if** (op.size() == 0) {

                                    System.*out*.println("No fruit found");

                        }

                        String s[] = **new** String[op.size()];

                        op.toArray(s);

**return** s;

            }

}

**143. Proper Case**

Write a program to read a string and convert the intial letter of each word to uppercase.

Include a class UserMainCode with a static method changeCase which accepts the string. The return type is the modified string.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

This is cognizant academy

**Sample Output 1:**

This Is Cognizant Academy

**public** **class** UserMainCode {

**public** **static** String properCase(String s1) {

                        StringBuffer sb = **new** StringBuffer();

                        String[] strArr = s1.split(" ");

**for** (String str : strArr) {

**char**[] stringArray = str.trim().toCharArray();

                    stringArray[0] = Character.*toUpperCase*(stringArray[0]);

                    str = **new** String(stringArray);

                    sb.append(str).append(" ");

                }

**return** sb.toString().trim();

            }

}

**144. Elements in ArrayList**

Use Collection Methods.

Write a program that takes two ArrayLists as input and  finds out all elements present either in A or B, but not in both.

Include a class UserMainCode with the static method arrayListSubtractor which accepts the two arraylists and returns an array.

Create a Class Main which would be used to read the inputs and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer (m) denoting the size of first arraylist. The next m elements would be the values of the first arraylist. The next input would be n denoting the size of the second arraylist. The next n elements would be the values of the second arraylist.

Output consists of an array. The elements in the output array need to be printed in sorted order.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

1

8

3

5

2

3

5

**Sample Output 1:**

1

8

**Sample Input 2:**

4

9

1

3

5

4

1

3

5

6

**Sample Output 2:**

6

9

**public** **class** UserMainCode {

**public** **static** **int**[] arrayListSubtrator(ArrayList<Integer> a,

                                    ArrayList<Integer> b) {

                        ArrayList<Integer> op = **new** ArrayList<Integer>();

                        op.addAll(a);

                        op.removeAll(b);

                        b.removeAll(a);

                        op.addAll(b);

**int** x[] = **new** **int**[op.size()];

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

                                    x[i] = op.get(i);

                        }

**return** x;

            }

}

**145. Unique Number**

Write a program that accepts an Integer as input and finds whether the number is Unique or not. Print Unique if the number is ?Unique?, else print ?Not Unique?.

Note: A Unique number is a positive integer (without leading zeros) with no duplicate digits.For example 7, 135, 214 are all unique numbers whereas 33, 3121, 300 are not.

Include a class UserMainCode with a static method getUnique which accepts an integer. The return type (Integer) should return 1 if the number is unique else return -1.

Create a Class Main which would be used to accept Input Integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer .

Output consists of a String (?Unique? or ?Not Unique?).

Refer sample output for formatting specifications.

**Sample Input 1:**

123

**Sample Output 1:**

Unique

**Sample Input 2:**

33

**Sample Output 2:**

Not Unique

**public** **class** UserMainCode {

**public** **static** **int** getUnique(**int** n) {

**int** flag = 1;

**int** c;

**int** no = n;

**for** (**int** i = 0; i <= 9; i++) {

                                    n = no;

                                    c = 0;

**while** (n > 0) {

**int** d = n % 10;

**if** (d == i)

                                                            c++;

                                                n /= 10;

                                    }

**if** (c > 1)

                                                flag = 0;

                        }

**if** (flag == 1) {

**return** 1;

                        } **else** {

**return** -1;

                        }

            }

}

**146. Length of same word**

Write a program to read a string containing multiple words find the first and last words, if they are same, return the length and if not return the sum of length of the two words.

Include a class UserMainCode with a static method compareLastWords which accepts the string. The return type is the length as per problem.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

This is Cognizant Academy

**Sample Output 1:**

11

**Sample Input 2:**

Hello World Hello

**Sample Output 2:**

5

**public** **class** UserMainCode {

**public** **static** **int** comparelastWord(String s1) {

                        String s2[] = s1.split(" ");

**if** (s2[0].equals(s2[(s2.length) - 1])) {

**return** s2[0].length();

                        } **else** {

**return** (s2[0].length() + s2[(s2.length) - 1].length());

                        }

            }

}

**148. Sum of Lowest marks**

Given input as HashMap, value consists of marks and rollno as key.Find the sum of the lowest three subject marks from the HashMap

Include a class UserMainCode with a static method getLowest which accepts a Hashmap with marks and rollno.

The return type of the output is the sum of lowest three subject marks.

Create a class Main which would get the input and call the static method getLowest present in the UserMainCode.

Input and Output Format:

First line of the input corresponds to the HashMap size.

Input consists a HashMap with marks and rollno.

Output is an integer which is the sum of lowest three subject marks.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

1

54

2

85

3

74

4

59

5

57

**Sample Output 1:**

170

**Sample Input 2:**

4

10

56

20

58

30

87

40

54

**Sample Output 2:**

168

**public** **class** UserMainCode {

**public** **static** **int** getLowestSum(HashMap<Integer, Integer> hm, **int** n) {

**int** sum = 0;

                        ArrayList<Integer> al = **new** ArrayList<Integer>();

                        Iterator<Integer> itr = hm.values().iterator();

**while** (itr.hasNext()) {

                                    al.add(itr.next());

                        }

                        Collections.*sort*(al);

**for** (**int** i = 0; i < 3; i++) {

                                    sum = sum + al.get(i);

                        }

**return** sum;

            }

}

**149. Perfect Number**

Write a program to that takes a positive integer and returns true if the number is perfect number.

A positive integer is called a perfect number if the sum of all its factors (excluding the number itself, i.e., proper divisor) is equal to its value.

For example, the number 6 is perfect because its proper divisors are 1, 2, and 3, and 6=1+2+3; but the number 10 is not perfect because its proper divisors are 1, 2, and 5, and 1+2+5 is not equal to 10

Include a class UserMainCode with a static method getPerfection which accepts the number. The return type is boolean (true / false).

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a integer.

Output consists of TRUE / FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

28

**Sample Output 1:**

TRUE

**public** **class** UserMainCode {

**public** **static** **boolean** perfectNumber(**int** n) {

**int** sum = 0;

**for** (**int** i = 1; i < n; i++) {

**if** (n % i == 0) {

                                                sum = sum + i;

                                    }

                        }

**if** (sum == n) {

**return** **true**;

                        } **else** {

**return** **false**;

                        }

            }

}

**150. Price Calculator - II**

Write a small price calculator application with the below mentioned flow:

1. Read a value n indicating the total count of devices. This would be followed by the name and price of the device. The datatype for name would be String and price would be float.

2. Build a hashmap containing the peripheral devices with name as key and price as value.

3. Read a value m indicating the number of devices for which the price has to be calculated. This would be followed by device names.

4. For each devices mentioned in the array calcuate the total price.

5. You decide to write a function costEstimator which takes the above hashmap and array as input and returns the total price (float) as output with two decimal points. Include this function in class UserMainCode.

Create a Class Main which would be used to read details in step 1 and build the hashmap. Call the static method present in UserMainCode.

Input and Output Format:

Input consists of device details. The first number indicates the size of the devices. The next two values indicate the name,price.

This would be followed by m indicating the size of the device array. The next m values would be the device names.

Output consists of the total price in float.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

Monitor

1200.36

Mouse

100.42

Speakers

500.25

2

Speakers

Mouse

**Sample Output 1:**

600.67

**public** **class** UserMainCode {

**public** **static** Float priceCalculator(HashMap<String, String> hm, String[] s1) {

                        Float f = (**float**) 0;

                        Iterator<String> it = hm.keySet().iterator();

**while** (it.hasNext()) {

                                    String s = (String) it.next();

                                    Float f1 = Float.*parseFloat*(hm.get(s));

**for** (**int** i = 0; i < s1.length; i++) {

**if** (s1[i].equals(s))

                                                            f += f1;

                                    }

                        }

**return** f;

            }

}

**151. Color Code Validation**

Give a String as colour code as input and write code to validate whether the given string is a valid color code or not.

**Validation Rule:**

String should start with the Character '#'.

Length of String is 7.

It should contain 6 Characters after '#' Symbol.

It should contain Characters between 'A-F' and Digits '0-9'.

If String acceptable the return true otherwise false.

Include a class UserMainCode with a static method validateColourCode which accepts a string as input.

The return type of the output is a boolean which returns true if its is a valid color code else it returns false

Create a class Main which would get the input and call the static method validateColourCode present in the UserMainCode.

Input and Output Format:

Input consists a string corresponding to the color code.

Output is a boolean which returns true or false

Refer sample output for formatting specifications.

**Sample Input 1:**

#99FF33

**Sample Output 1:**

true

**Sample Input 2:**

#CCCC99#

**Sample Output 2:**

false

**public** **class** UserMainCode {

**public** **static** **boolean** validateColorCode(String s1) {

**if** (s1.matches("[#]{1}[A-F0-9]{6}")) {

**return** **true**;

                        } **else** {

**return** **false**;

                        }

            }

}

**152. String Processing - ZigZag**

Write a program to read a string containing date in DD-MM-YYYY format. find the number of days in the given month.

Note - In leap year February has got 29 days.

Include a class UserMainCode with a static method getLastDayOfMonth which accepts the string. The return type is the integer having number of days.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

12-06-2012

**Sample Output 1:**

30

**Sample Input 2:**

10-02-2012

**Sample Output 2:**

29

**public** **class** UserMainCode {

**public** **static** **int** ZigZag(String s1) {

**int** n = 0;

                        SimpleDateFormat sdf = **new** SimpleDateFormat("dd-MM-yyyy");

                        Calendar cal = Calendar.*getInstance*();

**try** {

                                    Date d = sdf.parse(s1);

                                    cal.setTime(d);

                                    n = cal.getActualMaximum(Calendar.*DAY\_OF\_MONTH*);

                        } **catch** (Exception e) {

                                    e.printStackTrace();

                        }

**return** n;

            }

}

**153. String Processing – III(moveX)**

Write a program to read a string where all the lowercase 'x' chars have been moved to the end of the string.

Include a class UserMainCode with a static method moveX which accepts the string. The return type is the modified string.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string.

Refer sample output for formatting specifications.

Sample Input 1:

xxhixx

Sample Output 1:

hixxxx

Sample Input 2:

XXxxtest

Sample Output 2:

XXtestxx

**public** **class** UserMainCode {

**public** **static** String moveX(String s1) {

                        String s2 = s1.replaceAll("[x]", "");

                        String s3 = s1.replaceAll("[^x]", "");

                        String res = s2 + s3;

**return** res;

            }

**154. Leap Year**

Write a program to read a string containing date in DD/MM/YYYY format and check if its a leap year. If so, return true else return false.

Include a class UserMainCode with a static method isLeapYear which accepts the string. The return type is the boolean indicating TRUE / FALSE.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of TRUE / FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

23/02/2012

**Sample Output 1:**

TRUE

**Sample Input 2:**

12/12/2011

**Sample Output 2:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** isLeapYear(String s) {

**boolean** b = **false**;

                        String[] s1 = s.split("/");

**int** year = Integer.*valueOf*(s1[2]);

                        GregorianCalendar gc = **new** GregorianCalendar();

                        b = gc.isLeapYear(year);

**return** b;

            }

}

**155. String Processing – IV(Form New Word)**

Write a program to read a string and also a number N. Form a new string starting with 1st character and with every Nth character of the given string. Ex - if N is 3, use chars 1, 3, 6, ... and so on to form the new String. Assume N>=1.

Include a class UserMainCode with a static method getStringUsingNthCharacter which accepts the string and the number n. The return type is the string as per the problem statement.

Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string and integer.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

HelloWorld

2

**Sample Output 1:**

HelWrd

**public** **class** UserMainCode {

**public** **static** String getString(String s1, **int** n) {

                        String[] str=s1.split("");

                        StringBuffer sb=**new** StringBuffer();

                        sb.append(str[1]);

**for** (**int** i = 1; i < str.length; i++) {

**if**(i%n==0)

                                    {

                                                sb.append(str[i]);

                                    }

                        }

**return** sb.toString();

            }

}

**156. Repeating set of characters in a string**

Get a string and a positive integer n as input .The last n characters should repeat the number of times given as second input.Write code to repeat the set of character from the given string.

Include a class UserMainCode with a static method getString which accepts a string and an integer n as input.

The return type of the output is a string with repeated n characters.

Create a class Main which would get the input and call the static method getString present in the UserMainCode.

Input and Output Format:

Input consists a string and a positive integer n.

Output is a string with repeated characters.

Refer sample output for formatting specifications.

**Sample Input 1:**

Cognizant

3

**Sample Output 1:**

Cognizantantantant

**Sample Input 2:**

myacademy

2

**Sample Output 2:**

myacademymymy

**public** **class** UserMainCode

{

**public** **static** String repeatingChar(String s1, **int** s2)

            {

**int** s3 = s1.length();

                        String s4 = s1.substring(s3 - s2);

                        StringBuffer sb = **new** StringBuffer(s1);

**for** (**int** i = 0; i < s2; i++)

                        {

                                    sb.append(s4);

                        }

**return** sb.toString();

            }

}

**157. Largest Chunk**

Write a program to read a string and return the length of the largest "chunk" in the string.

A chunk is a repetition of same character 2 or more number of times. If the given string doest not contain any repeated chunk of characters return -1.

Include a class UserMainCode with a static method getLargestSpan which accepts the string. The return type is the integer.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

This place is soooo good

**Sample Output 1:**

4

**public** **class** UserMainCode {

**public** **static** **int** getlargestChunk(String s1) {

                        StringTokenizer st = **new** StringTokenizer(s1, "");

**int** max = 0;

**while** (st.hasMoreTokens()) {

                                    String s = st.nextToken();

                                    StringBuffer sb = **new** StringBuffer(s);

**for** (**int** i = 0; i < sb.length(); i++) {

**int** count = 0;

**for** (**int** j = i + 1; j < sb.length(); j++) {

**if** (sb.charAt(i) == sb.charAt(j)) {

                                                                        count++;

                                                            } **else** {

**break**;

                                                            }

                                                }

**if** (count > max) {

                                                            max = count + 1;

                                                }

                                    }

                        }

**if** (max == 0) {

**return** -1;

                        } **else** {

**return** max;

                        }

            }

}

**158. Digit Comparison**

Write a program to read two integers and return true if they have the same last digit.

Include a class UserMainCode with a static method compareLastDigit which accepts two integers and returns boolean. (true / false)

Create a Class Main which would be used to accept two integers and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two integer.

Output consists TRUE / FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

59

29

**Sample Output 1:**

TRUE

**public** **class** UserMainCode {

**public** **static** **boolean** digitComparison(**int** a, **int** b) {

**boolean** x = **false**;

**int** c = a % 10;

**int** d = b % 10;

**if** (c == d) {

                                    x = **true**;

                        } **else** {

**return** x = **false**;

                        }

**return** x;

            }

}

**160. Largest Span**

Write a program to read a integer array, find the largest span in the array.

Span is the count of all the elements between two repeating elements including the repeated elements.

Include a class UserMainCode with a static method getLargestSpan which accepts the integer array. The return type is integer.

Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n which is the number of elements followed by n integer values.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

6

4

2

1

4

5

7

**Sample Output 1:**

4

**public** **class** UserMainCode {

**public** **static** **int** display(**int**[] a, **int** n) {

**int** gap = 0, max = 0;

**for** (**int** i = 0; i < n; i++) {

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

**if** (a[i] == a[j]) {

                                                            gap = j;

                                                }

                                    }

**if** (gap - 1 > max)

                                                max = gap - i;

                        }

**return** max + 1;

            }

}

**162. Duplicates**

GIven three integers (a,b,c) find the sum. However, if one of the values is the same as another, both the numbers do not count towards the sum and the third number is returned as the sum.

Include a class UserMainCode with a static method getDistinctSum which accepts three integers and returns integer.

Create a Class Main which would be used to accept three integers and call the static method present in UserMainCode.

Input and Output Format:

Input consists of three integers.

Output consists of a integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

1

2

1

**Sample Output 1:**

2

**Sample Input 2:**

1

2

3

**Sample Output 2:**

6

**public** **class** UserMainCode {

**public** **static** **int** Duplicates(**int** a, **int** b, **int** c) {

**int** sum = 0;

**if** (a == b) {

                                    sum = c;

                        } **else** **if** (c == a) {

                                    sum = b;

                        } **else** **if** (b == c) {

                                    sum = a;

                        } **else** {

                                    sum = a + b + c;

                        }

**return** sum;

            }

}

**163. Even Sum & Duplicate Elements**

Write a program to read a integer array, Remove the duplicate elements and display sum of even numbers in the output. If input array contain only odd number then return -1.

Include a class UserMainCode with a static method sumElements which accepts the integer array. The return type is integer.

Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n which is the number of elements followed by n integer values.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

7

2

3

54

1

6

7

7

**Sample Output 1:**

62

**Sample Input 2:**

6

3

7

9

13

17

21

**Sample Output 2:**

-1

**public** **class** UserMainCode {

**public** **static** **int** sumElements(**int**[] a) {

**int** sum = 0, c = 0, op = 0;

**int** b[] = **new** **int**[a.length];

**int** s = a.length;

**for** (**int** i = 0; i < s; i++) {

**if** (a[i] % 2 != 0) {

                                                c++;

                                    }

                        }

**if** (c == s) {

                                    op = -1;

**return** op;

                        }

**for** (**int** i = 0; i < a.length; i++) {

**for** (**int** j = 0; j < a.length; j++) {

**if** (a[i] != a[j]) {

                                                            b[i] = a[i];

                                                }

                                    }

                        }

**for** (**int** i = 0; i < b.length; i++) {

**if** (b[i] % 2 == 0) {

                                                sum += b[i];

                                    }

                        }

**return** sum;

}

**164. String Processing - MixMania**

Write a program to read a string and check if it starts with '\_ix' where '\_' is any one char(a-z, A-Z, 0-9).

If specified pattern is found return true else false.

Include a class UserMainCode with a static method checkPattern which accepts the string. The return type is TRUE / FALSE.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string.

Output consists of TRUE / FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

Mix Mania

**Sample Output 1:**

TRUE

**public** **class** UserMainCode {

**public** **static** **boolean** checkPattern(String s) {

**if** (s.matches("[a-zA-Z0-9]{1}[i]{1}[x]{1}[\\w a-zA-z0-9]{1,}")) {

**return** **true**;

                        } **else** {

**return** **false**;

                        }

            }

}

**165. String Processing(First&LastChar)**

Write a program to read a string and return a new string where the first and last chars have been interchanged.

Include a class UserMainCode with a static method exchangeCharacters which accepts the string. The return type is the modified string.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string.

Output consists of string.

Refer sample output for formatting specifications.

**Sample Input 1:**

HelloWorld

**Sample Output 1:**

delloWorlH

**public** **static** String swapFirstLastChar(String s1) {

                        StringBuffer sb = **new** StringBuffer();

**char** c  =s1.charAt(0);

                        String s2 = s1.substring(1,s1.length()-1);

**char** c1 = s1.charAt(s1.length()-1);

                        sb.append(c1).append(s2).append(c);

**return** sb.toString();

            }

}

**166. Regular Expression – II(OnlyAlphabetString)**

Given a string (s) apply the following rules.

1. String consists of three characters only.

2. The characters should be alphabets only.

If all the conditions are satisifed then print TRUE else print FALSE.

Include a class UserMainCode with a static method validateString which accepts the string. The return type is the boolean formed based on rules.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string.

Output consists of TRUE or FALSE .

Refer sample output for formatting specifications.

**Sample Input 1:**

AcB

**Sample Output 1:**

TRUE

**Sample Input 2:**

A2B

**Sample Output 2:**

FALSE

Close

**public** **class** UserMainCode {

**public** **static** **boolean** validateString(String s1) {

**boolean** b=**false**;

**if**(s1.matches("[a-zA-Z]{3}"))

                        {

                                    b=**true**;

                        }

**return** b;

            }

}

**167. Strings Processing - Replication**

Write a program to read a string and also a number N. Return the replica of original string for n given time.

Include a class UserMainCode with a static method repeatString which accepts the the string and the number n. The return type is the string based on the problem statement.

Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string and integer.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Lily

2

**Sample Output 1:**

LilyLily

**public** **class** UserMainCode {

**public** **static** String stringRepetition(String s1, **int** n) {

                        StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < n; i++) {

                                    sb.append(s1);

                        }

**return** sb.toString();

            }

**168. Regular Expression – III(Validate Password)**

Given a string (s)  apply the following rules.

I)At least 8 characters must be present

II)At least one capital letter must be present

III)At least one small letter must be present

Iv)At least one special symbol must be present

V)At least one numeric value must be present

If the condition is satisifed then print valid else print invalid.

Include a class UserMainCode with a static method passwordValidation which accepts the string. The return type is the string.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string.

Output consists of string (valid / invalid) .

Refer sample output for formatting specifications.

**Sample Input 1:**

Technology$1213

**Sample Output 1:**

valid

**public** **class** UserMainCode {

**public** **static** String validatePassword(String s) {

                        String res = **null**;

**if** (s.matches("((?=.\*[0-9])(?=.\*[a-zA-Z])(?=.\*[@#$!]).{8,})")) {

                                    res = "Valid";

                        } **else** {

                                    res = "Invalid";

                        }

**return** res;

            }

}

**169. SumOdd**

Write a program to read an integer and find the sum of all odd numbers from 1 to the given number. [inclusive of the given number]

if N = 9 [ 1,3,5,7,9]. Sum = 25

Include a class UserMainCode with a static method addOddNumbers which accepts the number n. The return type is the integer based on the problem statement.

Create a Class Main which would be used to accept the integer and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a integer.

Output consists of a integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

6

**Sample Output 1:**

9

**public** **class** UserMainCode {

**public** **static** **int** addOddNumbers(**int** s) {

**int** sum = 0;

**for** (**int** i = 1; i <= s; i++) {

**if** (i % 2 != 0) {

                                                sum += i;

                                    }

                        }

**return** sum;

            }

}

**170. Integer Factorial**

Give an array of integer as input, store the numbers and their factorials in an hashmap and print the same.

Include a class UserMainCode with a static method getFactorial which accepts the integer array. The return type is the hashmap which is printed key:value.

Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a number denoting the size of the array and followed by the elements.

Output consists of a hashmap printed in the output format .

Refer sample output for formatting specifications.

**Sample Input1:**

4

2

3

5

4

**Sample Output1:**

2:2

3:6

5:120

4:24

**public** **class** UserMainCode {

**public** **static** LinkedHashMap<Integer, Integer> integerFactorial(**int**[] a) {

                        LinkedHashMap<Integer, Integer> hm = **new** LinkedHashMap<Integer, Integer>();

**for** (**int** i = 0; i < a.length; i++) {

**int** u=1;

**for** (**int** j = 1; j <= a[i]; j++) {

                                                u=u\*j;

                                                hm.put(a[i], u);

                                    }

                        }

**return** hm;

            }

}

**171. String Processing – V(concatString)**

Write a program to read a string array, concatenate the array elements one by one separated by comma and return the final string as output.

Include a class UserMainCode with a static method concatString which accepts the string array. The return type is the string.

Create a Class Main which would be used to accept the string array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n which is the number of elements followed by n string values.

Output consists of the string.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

AAA

BBB

CCC

**Sample Output 1:**

AAA,BBB,CCC

**public** **class** UserMainCode {

**public** **static** String concatString(String[] s1, **int** n) {

                        StringBuffer res = **new** StringBuffer();

**for** (**int** i = 0; i < s1.length; i++)

                                    res.append(s1[i]).append(",");

                        res.deleteCharAt(res.length() - 1);

**return** res.toString();

            }

}

**172. String processing ? Long + Short + Long**

Obtain two strings S1,S2 from user as input. Your program should form a string of  ?long+short+long?, with the shorter string inside of the longer String.

Include a class UserMainCode with a static method getCombo which accepts two string variables. The return type is the string.

Create a Class Main which would be used to accept two Input strings and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two strings with maximum size of 100 characters.

Output consists of an string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Hello

Hi

**Sample Output 1:**

HelloHiHello

**public** **class** UserMainCode {

**public** **static** String getCombo(String s1, String s2) {

                        StringBuffer sb1 = **new** StringBuffer();

**int** len1 = s1.length();

**int** len2 = s2.length();

**if** (len1 > len2) {

                                    sb1.append(s1).append(s2).append(s1);

                        } **else** {

                                    sb1.append(s2).append(s1).append(s2);

                        }

**return** sb1.toString();

            }

}

**173. Unique Number**

GIven three integers (a,b,c) , Write a program that returns the number of unique integers among the three.

Include a class UserMainCode with a static method calculateUnique which accepts three integers and returns the count as integer.

Create a Class Main which would be used to accept three integers and call the static method present in UserMainCode.

Input and Output Format:

Input consists of three integers.

Output consists of a integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

12

4

3

**Sample Output 1:**

3

**Sample Input 2:**

4

-4

4

**Sample Output 2:**

2

**public** **class** UserMainCode {

**public** **static** **int** uniqueNumberII(**int** a, **int** b, **int** c) {

**int** d = 0;

**if** (a != b && a != c) {

                                    d = 3;

                        } **else** **if** (a == b && a == c) {

                                    d = 1;

                        } **else** **if** (a != b && a == c) {

                                    d = 2;

                        } **else** **if** (a == b && a != c) {

                                    d = 2;

                        }

**return** d;

            }

}

**174. Math Calculator**

Write a program that accepts three inputs, first two inputs are operands in int form and third one being one of the following five operators: +, -, \*, /, %. Implement calculator logic and return the result of the given inputs as per the operator provided. In case of division, Assume the result would be integer.

Include a class UserMainCode with a static method calculator which accepts two integers, one operand and returns the integer.

Create a Class Main which would be used to accept three integers and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two integers and a character.

Output consists of a integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

23

2

\*

**Sample Output 1:**

46

**public** **class** UserMainCode {

**public** **static** **int** mathCalculator(**int** a, **int** b, **char** c) {

**int** res = 0;

**if** (c == '+') {

                                    res = a + b;

                        } **else** **if** (c == '-') {

                                    res = a - b;

                        } **else** **if** (c == '\*') {

                                    res = a \* b;

                        } **else** **if** (c == '/') {

                                    res = a / b;

                        } **else** **if** (c == '%') {

                                    res = a % b;

                        }

**return** res;

            }

}

**175. Age for Voting**

Given a date of birth (dd/MM/yyyy) of a person in string, compute his age as of 01/01/2015.

If his age is greater than 18, then println eligible else println not-eligible.

Include a class UserMainCode with a static method getAge which accepts the string value. The return type is the string.

Create a Class Main which would be used to accept the two string values and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two string.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

16/11/1991

**Sample Output 1:**

eligible

**public** **class** UserMainCode {

**public** **static** String ageForVoting(String s1) {

                        String now = "01/10/2015";

                        SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/yyyy");

                        Date date = **new** Date();

                        Date date2 = **new** Date();

                        String result = "";

**try** {

                                    date = sdf.parse(now);

                                    date2 = sdf.parse(s1);

                                    Calendar calendar = Calendar.*getInstance*();

                                    calendar.setTime(date);

                                    calendar.add(Calendar.*YEAR*, -18);

                                    Calendar calendar2 = Calendar.*getInstance*();

                                    calendar2.setTime(date2);

**if** (calendar2.before(calendar)) {

                                                result = "eligible";

                                    } **else** {

                                                result = "Not eligible";

                                    }

                        } **catch** (ParseException e) {

                                    e.printStackTrace();

                        }

**return** result;

            }

}

**176. Scores**

Write a program to read a integer array of scores, if 100 appears at two consecutive locations return true else return false.

Include a class UserMainCode with a static method checkScores which accepts the integer array. The return type is boolean.

Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n which is the number of elements followed by n integer values.

Output consists of a string that is either 'TRUE' or 'FALSE'.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

1

100

100

**Sample Output 1:**

TRUE

**Sample Input 2:**

3

100

1

100

**Sample Output 2:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** scores(**int** a[]) {

**boolean** b = **false**;

**for** (**int** i = 0; i < a.length - 1; i++) {

**if** (a[i] == 100 && a[i + 1] == 100) {

                                                b = **true**;

**break**;

                                    } **else** {

                                                b = **false**;

                                    }

                        }

**return** b;

            }

}

**177. ArrayFront**

Write a program to read a integer array and return true if one of the first 4 elements in the array is 9 else return false.

Note: The array length may be less than 4.

Include a class UserMainCode with a static method scanArray which accepts the integer array. The return type is true / false.

Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n which is the number of elements followed by n integer values.

Output consists of TRUE / FALSE.

Refer sample output for formatting specifications.

**Sample Input 1:**

6

1

2

3

4

5

6

**Sample Output 1:**

FALSE

**Sample Input 2:**

3

1

2

9

**Sample Output 2:**

TRUE

**public** **class** UserMainCode {

**public** **static** **boolean** scanArray(**int** a[]) {

**if** (a[0] == 9 || a[1] == 9 || a[2] == 9 || a[3] == 9) {

**return** **true**;

                        } **else** {

**return** **false**;

                        }

            }

**178. Word Count**

Given a string array (s) and non negative integer (n) and return the number of elements in the array which have same number of characters as the givent int N.

Include a class UserMainCode with a static method countWord which accepts the string array and integer. The return type is the string formed based on rules.

Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a an integer indicating the number of elements in the string array followed the elements and ended by the non-negative integer (N).

Output consists of a integer .

Refer sample output for formatting specifications.

**Sample Input 1:**

4

a

bb

b

ccc

1

**Sample Output 1:**

2

**Sample Input 2:**

5

dog

cat

monkey

bear

fox

3

**Sample Output 2:**

3

**public** **class** UserMainCode {

**public** **static** **int** countWord(String[] s, **int** a) {

**int** count = 0;

**for** (**int** i = 0; i < s.length; i++) {

**if** (s[i].length() == a) {

                                                count++;

                                    }

                        }

**return** count;

            }

}

**179. Finding the day of birth**

Given an input as date of birth of person, write a program to calculate on which day (MONDAY,TUESDAY....) he was born store and print the day in Upper Case letters.

Include a class UserMainCode with a static method calculateBornDay which accepts a string as input.

The return type of the output is a string which should be the day in which the person was born.

Create a class Main which would get the input and call the static method calculateBornDay present in the UserMainCode.

Input and Output Format:

NOTE: date format should be(dd-MM-yyyy)

Input consists a date string.

Output is a string which the day in which the person was born.

Refer sample output for formatting specifications.

**Sample Input 1:**

29-07-2013

**Sample Output 1:**

MONDAY

**Sample Input 2:**

14-12-1992

**Sample Output 2:**

MONDAY

**public** **class** UserMainCode {

**public** **static** String findDayofBirth(String s1) {

                        String s = "";

                        SimpleDateFormat sdf1 = **new** SimpleDateFormat("dd-MM-yyyy");

                        SimpleDateFormat sdf2 = **new** SimpleDateFormat("EEEEE");

**try**

                        {

                                    Date d = sdf1.parse(s1);

                                    s = sdf2.format(d);

                        }

**catch** (Exception e) {

                                    e.printStackTrace();

                        }

**return** s.toUpperCase();

            }

}

**181. Removing elements from HashMap**

Given a HashMap as input, write a program to perform the following operation :  If the keys are divisible by 3 then remove that key and its values and print the number of remaining keys in the hashmap.

Include a class UserMainCode with a static method afterDelete which accepts a HashMap as input.

The return type of the output is an integer which represents the count of remaining elements in the hashmap.

Create a class Main which would get the input and call the static method afterDelete present in the UserMainCode.

Input and Output Format:

First input corresponds to the size of hashmap

Input consists a HashMap

Output is an integer which is the count of remaining elements in the hashmap.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

339

RON

1010

JONS

3366

SMITH

2020

TIM

**Sample Output 1:**

2

**Sample Input 2:**

5

1010

C2WE

6252

XY4E

1212

M2ED

7070

S2M41ITH

8585

J410N

**Sample Output 2:**

3

**public** **class** UserMainCode {

**public** **static** **int** removingElementFromHashmap(HashMap<Integer, String> hm,

**int** n) {

**int** count = 0;

                        Iterator<Integer> itr = hm.keySet().iterator();

**while** (itr.hasNext()) {

**int** n1 = itr.next();

**if** (n1 % 3 != 0) {

                                                count++;

                                    }

                        }

**return** count;

            }

}

**182. Find Distance**

Write a Program that accepts four int inputs(x1,y1,x2,y2) as the coordinates of two points. Calculate the distance between the two points using the below formula.

Formula : square root of((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2))

Then, Round the result to return an int

Include a class UserMainCode with a static method findDistance which accepts four integers. The return type is integer representing the formula.

Create a Class Main which would be used to accept the input integers and call the static method present in UserMainCode.

Input and Output Format:

Input consists of four integers.

Output consists of a single integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

4

5

2

**Sample Output 1:**

3

**Sample Input 2:**

3

1

5

2

**Sample Output 2:**

2

**public** **class** UserMainCode {

**public** **static** **int** findDistance(**int** x1, **int** x2, **int** y1, **int** y2)

            {

**int** result = 0;

                        result= (**int**)Math.*round*(Math.*sqrt*((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2)));

**return** result;

            }

}

**183. Experience Calculator**

Write a program to read Date of Joining and current date as Strings and Experience as integer and validate whether the given experience and calculated experience are the same. Print ?true? if same, else ?false?.

Include a class UserMainCode with a static method calculateExperience which accepts 2 strings and an integer. The return type is boolean.

Create a Class Main which would be used to accept 2 string (dates) and an integer and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of 2 strings and an integer, where the 2 strings corresponds to the date of joining and current date, and the integer is the experience.

Output is either ?true? or ?false?.

Refer sample output for formatting specifications.

**Sample Input 1:**

11/01/2010

01/09/2014

4

**Sample Output 1:**

true

**Sample Input 2:**

11/06/2009

01/09/2014

4

**Sample Output 2:**

false

**public** **class** UserMainCode {

**public** **static** **boolean** calculateExperience(String s1, String s2, **int** n) {

**try** {

                                    SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/yyyy");

                                    sdf.setLenient(**false**);

                                    Date d1 = sdf.parse(s1);

                                    Calendar cal = Calendar.*getInstance*();

                                    cal.setTime(d1);

                                    Date d2 = cal.getTime();

                                    SimpleDateFormat sdf1 = **new** SimpleDateFormat("yyyy");

**int** y1 = Integer.*parseInt*(sdf1.format(d2));

                                    SimpleDateFormat sdf2 = **new** SimpleDateFormat("dd/MM/yyyy");

                                    sdf2.setLenient(**false**);

                                    Date d3 = sdf.parse(s2);

                                    Calendar cal1 = Calendar.*getInstance*();

                                    cal1.setTime(d3);

                                    Date d4 = cal1.getTime();

                                    SimpleDateFormat sdf3 = **new** SimpleDateFormat("yyyy");

**int** y2 = Integer.*parseInt*(sdf3.format(d4));

**int** exp = y2 - y1;

**if** (exp == n) {

**return** **true**;

                                    } **else** {

**return** **false**;

                                    }

                        } **catch** (ParseException e) {

                                    e.printStackTrace();

                        }

**return** **false**;

            }

}

**184. Word Count - II**

Write a program to read a string and count the number of words present in it.

Include a class UserMainCode with a static method countWord which accepts the string. The return type is the integer giving out the count of words.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

Today is Sunday

**Sample Output 1:**

3

**public** **class** UserMainCode

{

**public** **static** **int** countWord(String s1)

            {

                        String s[] = s1.split(" ");

**return** s.length;

            }

}

**185. Flush Characters**

Write a program to read a string from the user and remove all the alphabets and spaces from the String, and only store special characters and digit in the output String. Print the output string.

Include a class UserMainCode with a static method getSpecialChar which accepts a string. The return type (String) should return the character removed string.

Create a Class Main which would be used to accept a string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a strings.

Output consists of an String (character removed string).

Refer sample output for formatting specifications.

**Sample Input :**

cogniz$#45Ant

**Sample Output :**

$#45

**public** **class** UserMainCode

{

**public** **static** String getSpecialChar(String s1)

            {

                        String s2 = s1.replaceAll("[a-zA-Z \\s]","");

                        //s2.replaceAll(" +", "");

**return** s2;

            }

}

**186. Sum of Max & Min**

Write a Program that accepts three integers, and returns the sum of maximum and minimum numbers.

Include a class UserMainCode with a static method getSumMaxMin which accepts three integers. The return type is integer representing the formula.

Create a Class Main which would be used to accept the input integers and call the static method present in UserMainCode.

Input and Output Format:

Input consists of three integers.

Output consists of a single integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

12

17

19

**Sample Output 1:**

31

**public** **class** UserMainCode {

**public** **static** **int** sumMaxMain(**int** a, **int** b, **int** c) {

**int** d = 0;

**if** (a < b && b < c) {

                                    d = a + c;

                        } **else** **if** (a < b && b > c) {

                                    d = b + a;

                        } **else** **if** (a > b && b < c) {

                                    d = c + b;

                        }

**return** d;

            }

}

**187. Decimal to Binary Conversion**

Write a Program that accepts a decimal number n, and converts the number to binary.

Include a class UserMainCode with a static method convertDecimalToBinary which accepts an integer. The return type is long representing the binary number.

Create a Class Main which would be used to accept the input integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of single integer.

Output consists of a single long.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

**Sample Output 1:**

101

**public** **class** UserMainCode {

**public** **static** **long** converDecimalToBinary(**int** n) {

**int** decimal\_val = 0, base = 1, rem;

**while** (n > 0) {

                                    rem = n % 2;

                                    decimal\_val = decimal\_val + rem \* base;

                                    n = n / 2;

                                    base = base \* 10;

                        }

**return** decimal\_val;

            }

}

**188. String Repetition**

Write a program to read a string and an integer and return a string based on the below rules.

If input2 is equal or greater than 3 then repeat the first three character of the String by given input2 times, separated by a space.

If input2 is 2 then repeat the first two character of String two times separated by a space,

If input2 is 1 then return the first character of the String.

Include a class UserMainCode with a static method repeatString which takes a string & integer and returns a string based on the above rules.

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string and integer.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

COGNIZANT

4

**Sample Output 1:**

COG COG COG COG

**Sample Input 2:**

COGNIZANT

2

**Sample Output 2:**

CO CO

**public** **class** UserMainCode {

**public** **static** String repeatString(String a, **int** s) {

                        String b = " ";

                        StringBuffer buffer = **new** StringBuffer(a);

**if** (s >= 3) {

                                    b = buffer.substring(0, 3);

                        } **else** **if** (s <= 2) {

                                    b = buffer.substring(0, s);

                        }

**for** (**int** i = 1; i < s; i++) {

                                    System.*out*.print(b + " ");

                        }

**return** b;

            }

}

**189. String Processing – V(LastReapeatedCharacter)**

Write a program to read a string and also a number N. Form a new string made up of n repetitions of the last n characters of the String. You may assume that n is between 1 and the length of the string.

Include a class UserMainCode with a static method returnLastRepeatedCharacters which accepts the string and the number n. The return type is the string as per the problem statement.

Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string and integer.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Hello

2

**Sample Output 1:**

lolo

**Sample Input 2:**

Hello

3

**Sample Output 2:**

llollollo

**public** **class** UserMainCode {

**public** **static** String repeatString(String a, **int** s) {

                        StringBuffer buffer1 = **new** StringBuffer(a);

                        StringBuffer buffer2 = **new** StringBuffer();

                        buffer1.reverse();

**for** (**int** i = 0; i < s; i++) {

                                    buffer2.append(buffer1.substring(0, s));

                                    }

**return** buffer2.reverse().toString();

            }

}

**190. Average of Prime Locations**

Write a program to read an integer array and find the average of the numbers located on the Prime location(indexes).

Round the avarage to two decimal places.

Assume that the array starts with index 0.

Include a class UserMainCode with a static method averageElements which accepts a single integer array. The return type (double) should be the average.

Create a Class Main which would be used to accept Input array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of n+1 integers. The first integer corresponds to n, the number of elements in the array. The next 'n' integers correspond to the elements in the array.

Output consists of a single Double value.

Refer sample output for formatting specifications.

Assume that the maximum number of elements in the array is 20.

**Sample Input 1:**

8

4

1

7

6

5

8

6

9

**Sample Output 1:**

7.5

**public** **class** UserMainCode {

**public** **static** **double** averageElements(**int** a[]) {

**int** flag = 0, sum = 0, i;

**int** n = a.length;

**double** tot = 0, count = 0.0;

**for** (i = 0; i < n; i++) {

                                    flag = 0;

**for** (**int** j = 1; j <= i; j++) {

**if** (i % j == 0) {

                                                            flag++;

                                                }

                                    }

**if** (flag == 2) {

                                                sum = sum + a[i];

                                                count++;

                                    }

                        }

                        tot = (**double**) (sum / count);

**return** tot;

            }

}

**191. Regular Expression – III(NotBeginWithNumber)**

Given a string (s) apply the following rules.

1. String should not begin with a number.

If the condition is satisifed then print TRUE else print FALSE.

Include a class UserMainCode with a static method validateString which accepts the string. The return type is the boolean formed based on rules.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of TRUE or FALSE .

Refer sample output for formatting specifications.

**Sample Input 1:**

ab2

**Sample Output 1:**

TRUE

**Sample Input 2:**

72CAB

**Sample Output 2:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** beginString(String s1)

            {

**boolean** b=**false**;

**if**(s1.matches("[A-Za-z][a-zA-Z]\*[0-9]\*"))

                        {

                                    b=**true**;

                        }

**return** b;

            }

}

**192. Common Elements**

Write a program to read two integer arrays and find the sum of common elements in both the arrays. If there are no common elements return -1 as output

Include a class UserMainCode with a static method sumCommonElements which accepts two single integer array. The return type (integer) should be the sum of common elements.

Create a Class Main which would be used to accept Input array and call the static method present in UserMainCode.

Assume that all the elements will be distinct.

**Input and Output Format:**

Input consists of 2n+1 integers. The first integer corresponds to n, the number of elements in the array. The next 'n' integers correspond to the elements in the array, The last n elements correspond to the elements of the second array.

Output consists of a single Integer value.

Refer sample output for formatting specifications.

Assume that the maximum number of elements in the array is 20.

**Sample Input 1:**

4

1

2

3

4

2

3

6

7

**Sample Output 1:**

5

**public** **class** UserMainCode {

**public** **static** **int** commonElement(**int**[] a, **int**[] b) {

**int** sum = 0;

**for** (**int** i = 0; i < a.length; i++) {

**for** (**int** j = 0; j < b.length; j++) {

**if** (a[i] == b[j])

                                                            sum = sum + a[i];

                                    }

                        }

**if** (sum == 0)

**return** -1;

**else**

**return** sum;

            }

}

**193. String Processing - TrimCat**

Write a program to read a string and return a new string which is made of every alternate characters starting with the first character. For example NewYork will generate Nwok, and Samurai will generate Smri.

Include a class UserMainCode with a static method getAlternateChars which accepts the string. The return type is the modified string.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

**Input and Output Format:**

Input consists of a string.

Output consists of string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Hello

**Sample Output 1:**

Hlo

**public** **class** UserMainCode {

**public** **static** String trimCat(String s1) {

                        StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < s1.length(); i++) {

                                    sb.append(s1.charAt(i));

                                    i = i + 1;

                        }

                        String str = sb.toString();

**return** str;

            }

}

**194. Middle of Array**

Write a program to read an integer array and return the middle element in the array. The size of the array would always be odd.

Include a class UserMainCode with a static method getMiddleElement which accepts a single integer array. The return type (integer) should be the middle element in the array.

Create a Class Main which would be used to accept Input array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of n+1 integers. The first integer corresponds to n, the number of elements in the array. The next 'n' integers correspond to the elements in the array.

Output consists of a single Integer value.

Refer sample output for formatting specifications.

Assume that the maximum number of elements in the array is 19.

**Sample Input 1:**

5

1

5

23

64

9

**Sample Output 1:**

23

**public** **class** UserMainCode

{

**public** **static** **int** getMiddleElement(**int** a[])

            {

**int** count=a.length;

**return** a[count/2];

            }

}

**195. String Processing - Username**

Write a program to read a valid email id and extract the username.

Note - user name is the string appearing before @ symbol.

Include a class UserMainCode with a static method fetchUserName which accepts the string. The return type is the modified string.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of string.

Refer sample output for formatting specifications.

**Sample Input 1:**

admin@xyz.com

**Sample Output 1:**

admin

**public** **class** UserMainCode {

**public** **static** String fetchUserName(String s1) {

                        String[] st=s1.split("@");

                        String name = st[0];

**return** name;

            }

}

**196. String Processing - VII**

Write a program to read a two strings and one int value(N). check if Nth character of first String from start and Nth character of second String from end are same or not. If both are same return true else return false.

Check need not be Case sensitive

Include a class UserMainCode with a static method isEqual which accepts the two strings and a integer n. The return type is the TRUE / FALSE.

Create a Class Main which would be used to read the strings and integer and call the static method present in UserMainCode.

Input and Output Format:

Input consists of two strings and an integer.

Output consists of TRUE / FALSE .

Refer sample output for formatting specifications.

**Sample Input 1:**

AAAA

abab

2

**Sample Output 1:**

TRUE

**Sample Input 2:**

MNOP

QRST

3

**Sample Output 2:**

FALSE

**public** **class** UserMainCode {

**public** **static** **boolean** NthChar(String s1, String s2, **int** n) {

**boolean** b = **false**;

**char** c1 = s1.charAt(n);

**char** c2 = s2.charAt(n);

**if** (Character.*toString*(c1).equalsIgnoreCase(Character.*toString*(c2))) {

                                    b = **true**;

                        }

**return** b;

            }

}

**197. Simple String Manipulation**

Write a program to read a string and return a modified string based on the following rules.

Return the String without the first 2 chars except when

keep the first char if it is 'j'

keep the second char if it is 'b'.

Include a class UserMainCode with a static method getString which accepts a string. The return type (string) should be the modified string based on the above rules. Consider all letters in the input to be small case.

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string with maximum size of 100 characters.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

hello

**Sample Output 1:**

llo

**Sample Input 2:**

java

**Sample Output 2:**

jva

**public** **class** UserMainCode {

**public** **static** String getString(String s) {

                        StringBuffer sb = **new** StringBuffer();

                        // String output;

**if** (s.charAt(0) == 'j') {

**if** (s.charAt(1) == 'b') {

                                                sb.append(s);

                                    } **else** {

                                                sb.append(s.charAt(0)).append(s.substring(2));

                                    }

                        } **else** **if** (s.charAt(1) == 'b') {

                                    sb.append(s.charAt(0)).append(s.substring(2));

                        } **else** {

                                    sb.append(s.substring(2));

                        }

**return** sb.toString();

            }

}

**198. Largest Difference**

Write a program to read a integer array, find the largest difference between adjacent elements and display the index of largest difference.

EXAMPLE:

input1: {2,4,5,1,9,3,8}

output1: 4 (here largest difference 9-1=8 then return index of 9 ie,4)

Include a class UserMainCode with a static method checkDifference which accepts the integer array. The return type is integer.

Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.

Input and Output Format:

Input consists of an integer n which is the number of elements followed by n integer values.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

7

2

4

5

1

9

3

8

**Sample Output 1:**

4

**public** **class** UserMainCode {

**public** **static** **int** checkDiff(**int** a[]) {

**int** max = 0, p = 0;

**for** (**int** i = 0; i < a.length - 1; i++) {

**int** j = i + 1;

**int** t = a[i] - a[j];

**if** (t > max) {

                                                max = t;

                                                p = i;

                                    }

                        }

**return** p;

            }

}

**199. Date Validation**

Write a program to read a string representing a date. The date can be in any of the three formats

1:dd-MM-yyyy 2: dd/MM/yyyy 3: dd.MM.yyyy

If the date is valid, print valid else print invalid.

Include a class UserMainCode with a static method getValidDate which accepts a string. The return type (integer) should be based on the validity of the date.

Create a Class Main which would be used to accept Input string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string.

Refer sample output for formatting specifications.

**Sample Input 1:**

03.12.2013

**Sample Output 1:**

valid

**Sample Input 2:**

03$12$2013

**Sample Output 2:**

invalid

**public** **class** UserMainCode {

**public** **static** **int** dateValidate(String s1) {

**int** res = 0;

**if** (s1.matches("[0-9]{2}/[0-9]{2}/[0-9]{4}")) {

                                    SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/yyyy");

                                    sdf.setLenient(**false**);

**try** {

                                                Date d = sdf.parse(s1);

                                                res = 1;

                                    } **catch** (ParseException e) {

                                                res = 0;

                                    }

                        } **else** **if** (s1.matches("[0-9]{2}[.]{1}[0-9]{2}[.]{1}[0-9]{4}")) {

                                    SimpleDateFormat sdf1 = **new** SimpleDateFormat("dd.MM.yyyy");

                                    sdf1.setLenient(**false**);

**try** {

                                                Date d1 = sdf1.parse(s1);

                                                res = 1;

                                    } **catch** (ParseException e) {

                                                res = 0;

                                    }

                        } **else** **if** (s1.matches("[0-9]{2}[-]{1}[0-9]{2}[-]{1}[0-9]{4}")) {

                                    SimpleDateFormat sdf2 = **new** SimpleDateFormat("dd-MM-yyyy");

                                    sdf2.setLenient(**false**);

**try** {

                                                Date d2 = sdf2.parse(s1);

                                                res = 1;

                                    } **catch** (ParseException e) {

                                                res = 0;

                                    }

                        }

**return** res;

            }

}

**200. Boundary Average**

Given an int array as input, write a program to compute the average of the maximum and minimum element in the array.

Include a class UserMainCode with a static method ?getBoundaryAverage? that accepts an integer array as argument and returns a float that corresponds to the average of the maximum and minimum element in the array.

Create a class Main which would get the input array and call the static method getBoundaryAverage present in the UserMainCode.

Input and Output Format:

The first line of the input consists of an integer n, that corresponds to the size of the array.

The next n lines consist of integers that correspond to the elements in the array.

Assume that the maximum number of elements in the array is 10.

Output consists of a single float value that corresponds to the average of the max and min element in the array.

**Sample Input :**

6

3

6

9

4

2

5

**Sample Output:**

5.5

**public** **class** UserMainCode {

**public** **static** **float** boundaryAvg(**int** a[], **int** s) {

                        Arrays.*sort*(a);

**int** sum = a[0] + a[s - 1];

**float** avg = (**float**) sum / 2;

**return** avg;

            }

}

**201. Count Vowels**

Given a string input, write a program to find the total number of vowels in the given string.

Include a class UserMainCode with a static method ?countVowels? that accepts a String argument and returns an int that corresponds to the total number of vowels in the given string.

Create a class Main which would get the String as input and call the static method countVowels present in the UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of an integer..

**Sample Input:**

avinash

**Sample Output:**

3

**public** **class** UserMainCode

{

**public** **static** **int** countVowels(String s1)

            {

                        String s2 = s1.replaceAll("[aeiouAEIOU]", "");

**int** res = s1.length()-s2.length();

**return** res;

            }

**202. Month Name**

Given a date as a string input in the format dd-mm-yy, write a program to extract the month and to print the month name in upper case.

Include a class UserMainCode with a static method ?getMonthName? that accepts a String argument and returns a String that corresponds to the month name.

Create a class Main which would get the String as input and call the static method getMonthName present in the UserMainCode.

The month names are {JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER}

Input and Output Format:

Input consists of a String.

Output consists of a String.

**Sample Input:**

01-06-82

**Sample Output:**

JUNE

**public** **class** UserMainCode {

**public** **static** String getMonthname(String s1) {

                        String s2 = " ";

**try** {

                                    SimpleDateFormat sdf = **new** SimpleDateFormat("dd-MM-yy");

                                    sdf.setLenient(**false**);

                                    Date d1 = sdf.parse(s1);

                                    SimpleDateFormat sdf1 = **new** SimpleDateFormat("MMMM");

                                    s2 = sdf1.format(d1);

                        } **catch** (ParseException e) {

                                    e.printStackTrace();

                        }

**return** s2.toUpperCase();

            }

}

**203. Reverse SubString**

Given a string, startIndex and length, write a program to extract the substring from right to left. Assume the last character has index 0.

Include a class UserMainCode with a static method ?reverseSubstring? that accepts 3 arguments and returns a string. The 1st argument corresponds to the string, the second argument corresponds to the startIndex and the third argument corresponds to the length.

Create a class Main which would get a String and 2 integers as input and call the static method reverseSubstring present in the UserMainCode.

Input and Output Format:

The first line of the input consists of a string.

The second line of the input consists of an integer that corresponds to the startIndex.

The third line of the input consists of an integer that corresponds to the length of the substring.

**Sample Input:**

rajasthan

2

3

**Sample Output:**

hts

**public** **class** UserMainCode {

**public** **static** String reverseSubstring(String s1, **int** a, **int** b) {

                        StringBuffer sb = **new** StringBuffer(s1);

                        sb.reverse();

                        StringBuffer sb1 = **new** StringBuffer();

                        String ss = sb1.append(sb.substring(a, a + b)).toString();

**return** ss;

            }

}

**204. String Finder**

Given three strings say Searchstring, Str1 and Str2 as input, write a program to find out if Str2 comes after Str1 in the Searchstring.

Include a class UserMainCode with a static method ?stringFinder? that accepts 3 String arguments and returns an integer. The 3 arguments correspond to SearchString, Str1 and Str2. The function returns 1 if Str2 appears after Str1 in the Searchtring. Else it returns 2.

Create a class Main which would get 3 Strings as input and call the static method stringFinder present in the UserMainCode.

Input and Output Format:

Input consists of 3 strings.

The first input corresponds to the SearchString.

The second input corresponds to Str1.

The third input corresponds to Str2.

Output consists of a string that is either ?yes? or ?no?

**Sample Input 1:**

geniousRajKumarDev

Raj

Dev

**Sample Output 1:**

yes

**Sample Input 2:**

geniousRajKumarDev

Dev

Raj

**Sample Output 2:**

no

**public** **class** UserMainCode {

**public** **static** **int** stringFinder(String s1, String s2, String s3) {

**int** output = 0;

**if** (s1.contains(s2) && s1.contains(s3)) {

**if** (s1.indexOf(s2) < s1.indexOf(s3)) {

                                                output = 1;

                                    } **else**

                                                output = 2;

                        }

**return** output;

            }

}

**205. Phone Number Validator**

Given a phone number as a string input, write a program to verify whether the phone number is valid using the following business rules:

-It should contain only numbers or dashes (-)

- dashes may appear at any position

-Should have exactly 10 digits

Include a class UserMainCode with a static method ?validatePhoneNumber? that accepts a String input and returns a integer. The method returns 1 if the phone number is valid. Else it returns 2.

Create a class Main which would get a String as input and call the static method validatePhoneNumber present in the UserMainCode.

Input and Output Format:

Input consists of a string.

Output consists of a string that is either 'Valid' or 'Invalid'

**Sample Input 1:**

265-265-7777

**Sample Output 1:**

Valid

**Sample Input 2:**

265-65-7777

**Sample Output 1:**

Invalid

**public** **class** UserMainCode {

**public** **static** **int** phoneValidate(String s1) {

**if** (s1.matches("[0-9-]{12}")) {

**return** 1;

                        } **else** {

**return** 2;

                        }

            }

**206. Month : Number of Days**

Given two inputs year and month (Month is coded as: Jan=0, Feb=1 ,Mar=2 ...), write a program to find out total number of days in the given month for the given year.

Include a class UserMainCode with a static method ?getNumberOfDays? that accepts 2 integers as arguments and returns an integer. The first argument corresponds to the year and the second argument corresponds to the month code. The method returns an integer corresponding to the number of days in the month.

Create a class Main which would get 2 integers as input and call the static method getNumberOfDays present in the UserMainCode.

Input and Output Format:

Input consists of 2 integers that correspond to the year and month code.

Output consists of an integer that correspond to the number of days in the month in the given year.

**Sample Input:**

2000

1

**Sample Output:**

29

**public** **class** UserMainCode {

**public** **static** **int** getNumberOfDays(**int** year, **int** month) {

**int** days = 0;

**if** (month == 0 || month == 2 || month == 4 || month == 6 || month == 7

                                                || month == 9 || month == 11) {

                                    days = 31;

                        } **else** **if** (month == 3 || month == 5 || month == 8 || month == 10) {

                                    days = 30;

                        } **else** **if** (month == 1) {

                                    GregorianCalendar gc = **new** GregorianCalendar();

**boolean** b = gc.isLeapYear(year);

**if** (b == **true**) {

                                                days = 29;

                                    } **else** {

                                                days = 28;

                                    }

                        }

**return** days;

            }

}

**207. Negative String**

Given a string input, write a program to replace every appearance of the word "is" by "is not".

If the word "is" is immediately preceeded or followed by a letter no change should be made to the string .

Include a class UserMainCode with a static method ?negativeString? that accepts a String arguement and returns a String.

Create a class Main which would get a String as input and call the static method negativeString present in the UserMainCode.

Input and Output Format:

Input consists of a String.

Output consists of a String.

**Sample Input 1:**

This is just a misconception

**Sample Output 1:**

This is not just a misconception

**Sample Input 2:**

Today is misty

**Sample Output 2:**

Today is not misty

**public** **class** UserMainCode {

**public** **static** String negativeString(String s1) {

                        StringBuffer sb = **new** StringBuffer();

                        String s2[] = s1.split(" ");

**for** (**int** i = 0; i < s2.length; i++) {

**if** (s2[i].equals("is")) {

                                                sb.append(s2[i].replace("is", "is not")).append(" ");

                                    } **else** {

                                                sb.append(s2[i]).append(" ");

                                    }

                        }

**return** sb.deleteCharAt(sb.length() - 1).toString();

            }

}

**208. Validate Number**

Given a negative number as string input, write a program to validate the number and to print the corresponding positive number.

Include a class UserMainCode with a static method ?validateNumber? that accepts a string argument and returns a string. If the argument string contains a valid negative number, the method returns the corresponding positive number as a string. Else the method returns -1.

Create a class Main which would get a String as input and call the static method validateNumber present in the UserMainCode.

Input and Output Format:

Input consists of a String.

Output consists of a String.

**Sample Input 1:**

-94923

**Sample Output 1:**

94923

**Sample Input 2:**

-6t

**Sample Output 2:**

-1

**public** **class** UserMainCode {

**public** **static** String validateNumber(String s1) {

**int** s2;

                        String s3 = **null**;

**if** (s1.matches("[-0-9]{1,}")) {

                                    s2 = Math.*abs*(Integer.*parseInt*(s1));

                                    s3 = String.*valueOf*(s2);

                        } **else** {

**return** "-1";

                        }

**return** s3;

            }

}

**209. Digits**

Write a program to read a non-negative integer n, that returns the count of the occurances of 7 as digit.

Include a class UserMainCode with a static method countSeven which accepts the integer value. The return type is integer which is the count value.

Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.

Input and Output Format:

Input consists of a integer.

Output consists of integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

717

**Sample Output 1:**

2

**Sample Input 2:**

4534

**Sample Output 2:**

0

**public** **class** UserMainCode {

**public** **static** **int** digits(**int** s1) {

**int** rem = 0, count = 0;

**while** (s1 > 0) {

                                    rem = s1 % 10;

**if** (rem == 7)

                                                count++;

                                    s1 = s1 / 10;

                        }

**return** count;

            }

}​