**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 SumOfOddsAndEvens(int n){

int n1,n2=0,n3;

while(n!=0)

{

n1=n%10;

if((n1%2)!=0)

n2+=n1;

n/=10;

}

if(n2%2==0)

n3=-1;

else

n3=1;

return n3;

}

public static void main(String[] args) {

int n=84882;

System.out.println(SumOfOddsAndEvens(n));

}

}

**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

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner s=new Scanner(System.in);

String pan=s.next();

int b=panNumberValidation(pan);

if(b==1)

System.out.println("valid Pancard Number");

else

System.out.println("not a valid credential");

}

public static int panNumberValidation(String input) {

int b=0;

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

{b=1;}

else

b=0;

return b;

}

}

**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 display(int number){

int n1=0,n2=0;

while(number!=0)

{

n1=number%10;

if((n1%2)==0)

n2+=n1\*n1;

number/=10;

}

return n2;

}

}

**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

**import** java.util.Scanner;

**public** **class** Middle {

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

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

String s=sc.nextLine();

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

**if**(s.length()%2==0)

{

sb.append(s.substring(s.length()/2-1,s.length()/2+1));

//System.out.println(sb.toString());

}

System.*out*.println(sb.toString());

}

}

**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

**import** java.util.Scanner;

**public** **class** Main {

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

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

String s=sc.nextLine();

**int** res=UserMainCode.*checkCharacter*(s);

**if**(res==1)

{

System.*out*.println("Valid");

}

**else**

System.*out*.println("Invalid");

}

}

**public** **class** UserMainCode {

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

{

**int** res=-1;

**if**(s.charAt(0)==s.charAt(s.length()-1))

{

res=1;

}

**return** res;

}

}

**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

**import** java.util.Scanner;

**public** **class** Main {

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

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

String s=sc.nextLine();

**int** n=sc.nextInt();

System.*out*.println(UserMainCode.*stringChange*(s,n));

}

}

**public** **class** UserMainCode {

**public** **static** String stringChange(String s,**int** n)

{

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

sb.append(s.substring(0,n));

sb.append(s.substring(s.length()-n));

**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

**import** java.util.Scanner;

**public** **class** Main

{

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

{

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

**int** a=sc.nextInt();

System.*out*.println(UserMainCode.*reverse*(a));

}

}

**public** **class** UserMainCode

{

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

{

String s=String.*valueOf*(a);

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

sb.reverse(); //reverse return type is void

**int** res=Integer.*parseInt*(sb.toString());

**return** res;

}

}

**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**

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** Main {

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

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

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

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

**for**(**int** i=0;i<=4;i++) {

a.add(sc.nextInt());

}

**for**(**int** j=0;j<=4;j++) {

b.add(sc.nextInt());

}

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

c=Usermain.*sortMergedArrayList*(a,b);

**for**(**int** k=0;k<c.size();k++) {

System.***out***.println(c.get(k));

}

}

}

**import** java.lang.reflect.Array;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collections;

**public** **class** Usermain {

**public** **static** ArrayList<Integer> sortMergedArrayList(ArrayList<Integer> a, ArrayList<Integer> b) {

a.addAll(b);

Collections.*sort*(a);

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

c.add(a.get(2));

c.add(a.get(6));

c.add(a.get(8));

**return** c;

}

}

**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

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Scanner;

**public** **class** Qus8Main {

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

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

String s=sc.nextLine();

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

sdf.setLenient(**false**);

**int** res=0;

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

{

**try** {

Date d=sdf.parse(s);

res=1;

} **catch** (ParseException e) {

res=-1;

}

System.*out*.println(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

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Scanner;

**public** **class** Qus8Main {

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

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

String s=sc.nextLine();

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

sdf.setLenient(**false**);

**int** res=0;

**try** {

Date d=sdf.parse(s);

res=1;

} **catch** (ParseException e) {

res=-1;

}

System.*out*.println(res);

}

}

**11.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 Main {

public static void main(String[] args) {

String s1="zzzz";

System.out.println(stringFormatting(s1));

}

public static String stringFormatting(String s1) {

StringBuffer sb=new StringBuffer();

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

char c=s1.charAt(i);

if(i%2==0){

if(c==122)

c=(char) (c-25);

else{

c=(char) (c+1);}

sb.append(c);}

else

sb.append(c);}

return sb.toString();

}

}

12.Password Validation

Given a method with a password in string format as input. Write code to validate the password using following rules:

- Must contain at least one digit

- Must contain at least one of the following special characters @, #, $

# Length should be between 6 to 20 characters.

Include a class **UserMainCode** with a static method **validatePassword** which accepts a password string as input.

If the password is as per the given rules return 1 else return -1.If the return value is 1 then print valid password else print as invalid password.

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

**Input and Output Format:**

Input is a string .

Output is a string .

**Sample Input 1:**

%Dhoom%

**Sample Output 1:**  
Invalid password

**Sample Input 2:**

#@6Don

**Sample Output 2:**

Valid password

public class UserMainCode {

public static int display(String password){

if(password.matches(".\*[0-9]{1,}.\*") && password.matches(".\*[@#$]{1,}.\*") && password.length()>=6 && password.length()<=20)

{

return 1;

}

else

{

return -1;

}

}

}

13.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 Main {

public static void main(String[] args) {

String s1="capacity";

System.out.println(removeEvenElements(s1));

}

public static String removeEvenElements(String s1) {

StringBuffer sb1=new StringBuffer();

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

if((i%2)==0)

sb1.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')

sb1.append(s1.charAt(i));

return sb1.toString();

}

}

**14.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 useerm{

public static int display(int n,int[]a)

{

{

int sum=0;

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

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

return sum;

}}}

**15.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

import java.util.Arrays;

public class kape1 {

public static int display(int []array)

{

Arrays.sort(array);

int n=array[array.length-1]-array[0];

int b=array.length;

if(b==1)

{

n=array[0];

}

return n;

}

}

**16.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 array

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

**import** java.util.Scanner;

**public** **class** Main {

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

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

**int** n = sc.nextInt();

String[] sa = **new** String[n];

**for** (**int** i = 0; i < n; i++) {

sa[i] = sc.next();

}

String sr = sc.next();

System.***out***.println(Usermain.*getElementPosition*(sa, sr));

}

}

**import** java.awt.List;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collections;

**public** **class** Usermain {

**public** **static** **int** getElementPosition(String[] sa, String sr) {

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

**int** v = 0;

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

a.add(sa[i].toString());

}

Collections.*sort*(a, Collections.*reverseOrder*());

**for** (**int** l = 0; l < a.size(); l++) {

**if** (a.get(l).equals(sr)) {

v = l;

}

}

**return** v+1;

}

}

**17.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** generateSeries(**int** n)

{

**int** i=0,sumo=0,sume=0,sum=1;

**if**(n==1)

{

sum=n;

break;

}

**for**(i=3;i<=n;i=i+4)

{

sumo=sumo+i;

}

**for**(i=5;i<=n;i=i+4)

{

sume=sume+i;

}

sum+=sumo-sume;

**return** sum;

}

}

**18.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

**import** java.util.Scanner;

**public** **class** Main {

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

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

String s1=sc.nextLine();

String s2=sc.nextLine();

**int** c=sc.nextInt();

System.*out*.println(UserMainCode.*calculateElectricityBill*(s1,s2, c));

}

}

**public** **class** UserMainCode {

**public** **static** **int** calculateElectricityBill(String s1,String s2,**int** c)

{

**int** a=Integer.*parseInt*(s1.substring(5));

**int** b=Integer.*parseInt*(s2.substring(5));

**int** res=Math.abs((b-a)\*c);

**return** res;

}

}

**19.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 Main {

public static void main(String[] args) {

String s1="goodbad";

getvalues(s1);

}

public static void getvalues(String s1) {

int sum=0;

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

{

char a=s1.charAt(i);

if(Character.isDigit(a))

{

int b=Integer.parseInt(String.valueOf(a));

sum=sum+b;

}

}

if(sum==0)

{

System.out.println(-1);

}

else

System.out.println(sum);

}

}

**20.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

**import** java.util.Scanner;

**public** **class** Main {

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

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

String s1=sc.nextLine();

String s2=sc.nextLine();

System.*out*.println(UserMainCode.*concatString*(s1,s2));

}

}

**public** **class** UserMainCode

{

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

{

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

**int** n=s2.length();

sb.append(s1.substring(s1.length()-n));

sb.append(s2.substring(0));

**return** sb.toString();

}

}

**21.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

**import** java.util.Scanner;

**public** **class** Main {

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

{

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

String s=sc.nextLine();

//UserMainCode u=new UserMainCode();

**int** b=UserMainCode.*validateColorCode*(s);

**if**(b==1)

{

System.*out*.println("Valid color code");

}

**else**

{

System.*out*.println("Invalid color code ");

}

}

}

**public** **class** UserMainCode

{

**public** **static** **int** validateColorCode(String a)

{

**int** r=-1;

**if**(a.matches("(#)[A-F0-9]{6}"))

{

r=1;

}

**return** r;

}

}

**22.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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CTS-215

public class Main {

public static void main(String[] args) {

String s1="CTS-2j4";

getvalues(s1);

}

public static void getvalues(String s1) {

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

{

System.out.println(1);

}

else

System.out.println(-1);

}

}

**23.Removing Keys from HashMap**

Given a method with a HashMap<Integer,string> 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<integer,string>.

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

**24.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

**import** java.util.Scanner;

**public** **class** Main {

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

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

**int** n=sc.nextInt();

**int** a[]=**new** **int**[n];

**for**(**int** i=0;i<n;i++)

{

a[i]=sc.nextInt();

}

System.*out*.println(UserMainCode.*checkLargestAmongCorner*(a));

}

}

**public** **class** UserMainCode

{

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

{

**int** max=0;

**int** m=a[a.length/2];

**int** f=a[0];

**int** l=a[a.length-1];

**if**(m>f && m>l)

{

max=m;

}

**else** **if**(f>m && f>l)

{

max=f;

}

**else**

max=l;

**return** max;

}

}

**25.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

**import** java.util.Scanner;

**public** **class** Main {

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

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

**int** n=sc.nextInt();

**int** r=sc.nextInt();

System.*out*.println(UserMainCode.*calculateNcr*(n,r) );

}

}

**public** **class** UserMainCode

{

**public** **static** **int** calculateNcr(**int** n,**int** r)

{

**int** i,prod=1,prod1=1,prod2=1;

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

{

prod=prod\*i;

}

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

{

prod1=prod1\*i;

}

**int** diff=n-r;

**for**(i=1;i<=diff;i++)

{

prod2=prod2\*i;

}

**int** dem=prod1\*prod2;

**int** res=prod/dem;

**return** res;

}

}

**26.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 Main {**

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

**{**

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

**int n=sc.nextInt();**

**int m=sc.nextInt();**

**int[] a=new int[n];**

**int[] b=new int[m];**

**for(int i=0;i<n;i++)**

**a[i]=sc.nextInt();**

**for(int i=0;i<m;i++)**

**b[i]=sc.nextInt();**

**int u=UserMainCode.display(a,b);**

**if(u==-1)**

**System.out.println("No common elements");**

**else**

**System.out.println(u);}}**

**public class UserMainCode {**

**public static int display(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;**

**}}**

**27.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 int validatePassword(String password){

String regEx = "[^0-9|@|\_|#](.)\*[@|\_|#](.)\*[^@|\_|#]";

if(password.length()>=8 && password.matches(regEx)){

return 1;

}

return -1;

}

}

**if**(s.length()>=8&&s.matches("[^0-9|#|\_|@](.\*)[#|\_|@](.\*)[^@|\_|#]"))

{

System.*out*.println("valid");

}

**28.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

import java.util.\*;

public class Main {

public static void main(String[] args) {

String s1="CTS-hyd-1234";

String s2="hyderabad";

boolean b=formattingString(s1,s2);

if(b==true)

System.out.println("String format:CTS-LLL-XXXX// valid id");

else

System.out.println("not in required format");

}

public static boolean formattingString(String s1, String s2) {

String s3=s2.substring(0, 3);

boolean b=false;

StringTokenizer t=new StringTokenizer(s1,"-");

String s4=t.nextToken();

String s5=t.nextToken();

String s6=t.nextToken();

if(s4.equals("CTS") && s5.equals(s3) && s6.matches("[0-9]{4}"))

b=true;

else{

b=false;}

return b;

}

}

**29.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

import java.util.\*;

public class Main

{

public static void main(String[] args)

{

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

**int** n=sc.nextInt();

sc.nextLine();

String[] a=**new** String[n];

**for**(**int** i=0;i<n;i++)

a[i]=sc.nextLine();

**int** m=sc.nextInt();

System.out.println(UserMainCode.display(a,m));

}}

import java.util.\*;

public class UserMainCode

{

public static int display(String[] a,int m){

int u=a.length;

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

{

if(a[i].length()==m)

u--;

}

return u;

}}

**30.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

**import java.text.\*;**

**import java.util.\*;**

**public class Main {**

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

**String s1="2012-03-01";**

**String s2="2012-03-16";**

**System.out.println(monthsBetweenDates(s1,s2));**

**}**

**public static int monthsBetweenDates(String s1, String s2) throws ParseException {**

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

**Date d1=sdf.parse(s1);**

**Date d2=sdf.parse(s2);**

**Calendar cal=Calendar.getInstance();**

**cal.setTime(d1);**

**int months1=cal.get(Calendar.MONTH);**

**int year1=cal.get(Calendar.YEAR);**

**cal.setTime(d2);**

**int months2=cal.get(Calendar.MONTH);**

**int year2=cal.get(Calendar.YEAR);**

**int n=((year2-year1)\*12)+(months2-months1);**

**return n;**

**}**

**}**

**31.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 Main {

public static void main(String[] args) {

int a[]={2,4,3,5,6};

System.out.println(summationPattern(a));

}

public static int summationPattern(int[] a) {

int n1=0,n2=0;

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

if(a[i]%2==0)

n1+=(a[i]\*a[i]);

else

n2+=(a[i]\*a[i]\*a[i]);

return n1+n2;

}

}

**32.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

import java.util.\*;

public class Main {

public static void main(String[] args) {

String ipAddress="10.230.110.160";

boolean b=validateIpAddress(ipAddress);

if(b==true)

System.out.println("valid ipAddress");

else

System.out.println("not a valid ipAddress");

}

public static boolean validateIpAddress(String ipAddress) {

boolean b1=false;

StringTokenizer t=new StringTokenizer(ipAddress,".");

String s=t.nextToken();

**int** a=Integer.*parseInt*(s);

int b=Integer.parseInt(t.nextToken());

int c=Integer.parseInt(t.nextToken());

int d=Integer.parseInt(t.nextToken());

if((a>=0 && a<=255)&&(b>=0 && b<=255)&&(c>=0 && c<=255)&&(d>=0 && d<=255))

b1=true;

return b1;

}

}

**33.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

import java.text.\*;

import java.util.\*;

public class Main {

public static int dateDifference(String s1,String s2) throws ParseException{

SimpleDateFormat sd=new SimpleDateFormat("yyyy-MM-dd");

Date d=sd.parse(s1);

Calendar c=Calendar.getInstance();

c.setTime(d);

long d1=c.getTimeInMillis();

d=sd.parse(s2);

c.setTime(d);

long d2=c.getTimeInMillis();

int n=Math.abs((int) ((d1-d2)/(1000\*3600\*24)));

return n;

}

public static void main(String[] args) throws ParseException {

String s1="2012-03-12";

String s2="2012-03-14";

System.out.println(dateDifference(s1,s2));

}

}

**34.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

import java.util.\*;

public class Main {

public static String extensionString(String s1){

StringTokenizer t=new StringTokenizer(s1,".");

String ss=t.nextToken();

String s2=t.nextToken();

return s2;

}

public static void main(String[] args) {

String s1="sun.gif";

System.out.println(extensionString(s1));

}

}

**35.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

import java.util.Arrays;

import java.util.StringTokenizer;

public class PO

{

public static int display(String s,String s1)

{

int c=0,m=0;String t=null;

char a[]=s.toCharArray();

char b[]=s1.toCharArray();

Arrays.sort(a);

Arrays.sort(b);

s=new String(a);

s1=new String(b);

StringTokenizer st=new StringTokenizer(s);

StringTokenizer st1=new StringTokenizer(s1);

s=st.nextToken();

s1=st1.nextToken();

if(s.length()>s1.length())

{t=s1;

s1=s;

s=t;

}

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

{

for(int j=0;j<s1.length();j++)

{

if(s.charAt(i)==s1.charAt(j))

{

if((s.indexOf(s.charAt(i))==s.lastIndexOf(s.charAt(i)))&&(s1.indexOf(s1.charAt(j))==s1.lastIndexOf(s1.charAt(j))))

{

c++;

}

}}}

return c;

}

}

**36.)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

import java.util.StringTokenizer;

public class Main {

public static void main(String[] args) {

String s1="vishal jadiya";

getvalues(s1);

}

public static void getvalues(String s1) {

StringBuffer sb=new StringBuffer();

StringTokenizer st=new StringTokenizer(s1," ");

String s2=st.nextToken();

String s3=st.nextToken();

sb.append(s3);

sb.append(",");

sb.append(s2.substring(0,1).toUpperCase());

System.out.println(sb);

}

}

**37) 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

**import** java.util.Scanner;

**public** **class** Qus8Main {

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

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

String name=sc.nextLine();

**char** ch=sc.nextLine().charAt(0);

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

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

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

{

sb.deleteCharAt(i);

i--;

}

}

System.*out*.print(sb.toString());

}}

**38) 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

**import** java.util.Scanner;

**public** **class** Qus8Main {

**public** **static** **void** main(String[] name)

{

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

String s=sc.nextLine();

System.*out*.println(Qus8.*display*(s));

}

}

**public** **class** Qus8 {

**public** **static** **int** display(String name){

String s1=name;

**int** n1=0,n2=0,n3=0,n4=0,n5=0;

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

**char** c=s1.charAt(i);

**if**(c=='a' || c=='A')

n1++;

**if**(c=='e' || c=='E')

n2++;

**if**(c=='i' || c=='I')

n3++;

**if**(c=='o' || c=='O')

n4++;

**if**(c=='u' || c=='U')

n5++;}

**if**(n1==1 && n2==1 && n3==1 && n4==1 && n5==1)

**return** 1;

**else**

**return** 0 ;

}

}

**39) 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

**import** java.util.Scanner;

**public** **class** Qus8Main {

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

String s1="TRAINER";

*getvalues*(s1);

}

**public** **static** **void** 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)

sb.append(a);

}

System.*out*.println(sb);

}

**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));

System.*out*.println(sb);

}

}

}

**40) Average of Elements in Hashmap**

  Given a method with a HashMap<int, float> 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<int, float> 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<int, float>.

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

import java.util.HashMap;

import java.util.Scanner;

public class Main {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

HashMap<Integer, Float> hm = new HashMap<Integer, Float>();

int n = sc.nextInt();

for (int i = 0; i < n; i++) {

hm.put(sc.nextInt(), sc.nextFloat());

}

System.out.println(Usermain.avgOfEven(hm));

}

}

**import** java.lang.reflect.Array;

**import** java.math.BigDecimal;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Set;

**public** **class** Usermain {

**public** **static** Float avgOfEven(HashMap<Integer, Float> hm) {

Collection<Float> a = hm.values();

Set<Integer> l = hm.keySet();

ArrayList<Integer> m0 = **new** ArrayList<Integer>(l);

ArrayList<Float> m = **new** ArrayList<Float>(a);

Float c = (**float**) 0;

Float avg = (**float**) 0;

**int** p = 0;

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

**int** k = m0.get(i);

**if** (k % 2 == 0) {

Float b = m.get(i);

c = c + b;

p++;

}

}

avg = c / p;

Float k=(*round*(avg,2));//two decimal digits, //2.35

**return** k;

}

**public** **static** **float** round(**float** d, **int** decimalPlace)

**return** BigDecimal.*valueOf*(d).setScale(decimalPlace,BigDecimal.***ROUND\_HALF\_UP***).

floatValue();

}

}

**41)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<Integer,Double> 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

import java.util.HashMap;

import java.util.Scanner;

public class Main {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

HashMap<Integer, Float> hm = new HashMap<Integer, Float>();

int n = sc.nextInt();

for (int i = 0; i < n; i++) {

hm.put(sc.nextInt(), sc.nextFloat());

}

System.out.println(Usermain.avgOfEven(hm));

}

}

**import** java.lang.reflect.Array;

**import** java.math.BigDecimal;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Set;

**public** **class** Usermain {

**public** **static** Float avgOfEven(HashMap<Integer, Float> hm) {

Collection<Float> a = hm.values();

Set<Integer> l = hm.keySet();

ArrayList<Integer> m0 = **new** ArrayList<Integer>(l);

ArrayList<Float> m = **new** ArrayList<Float>(a);

Float c = (**float**) 0;

Float avg = (**float**) 0;

**int** p = 0;

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

**int** k = m0.get(i);

**if** (!(k % 2 == 0)) {

Float b = m.get(i);

c = c + b;

p++;

}

}

avg = c / p;

Float k=(*round*(avg,2));//two decimal digits, //2.35

**return** k;

}

**public** **static** **float** round(**float** d, **int** decimalPlace) {

**return** BigDecimal.*valueOf*(d).setScale(decimalPlace,BigDecimal.***ROUND\_HALF\_UP***).floatValue();

}

}

**42) Count Sequential Characters**

109.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 Main {

public static void main(String[] args) {

String input1="aaxxyzAAx";

System.out.println(consecutiveRepeatitionOfChar(input1));

}

public static int consecutiveRepeatitionOfChar(String input1) {

int c=0;

int n=0;

for(int i=0;i<input1.length()-1;i++){

if(input1.charAt(i)==input1.charAt(i+1))

n++;

else

n=0;

if(n==2)

c++; }

return c;

}

}

**43) 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

import java.util.\*;

public class Main {

public static void main(String[] args) {

String s1="You are toooo good";

System.out.println(maxChunk(s1));

}

public static int maxChunk(String s1) {

int max=0;

StringTokenizer t=new StringTokenizer(s1," ");

while(t.hasMoreTokens()){

String s2=t.nextToken();

int n=0;

for(int i=0;i<s2.length()-1;i++)

if(s2.charAt(i)==s2.charAt(i+1))

n++;

if(n>max)

max=n;

}

return (max+1);

}

}

**44) 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 Main {

public static void main(String[] args) {

String s1="HelloWorld";

getvalues(s1);

}

public static void getvalues(String s1) {

String s2=s1.toLowerCase();

StringBuffer sb=new StringBuffer(s2);

int l=sb.length();

int count=0;

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

{ count=0;

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

{

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

{

sb.deleteCharAt(j);

count++;

j--;

l--;

j=i;

}

}

if(count>0)

{

sb.deleteCharAt(i);

i--;

l--;

}

}

if(sb.length()==0)

{

System.out.println(-1);

}

else

System.out.println(sb.length());

}

}

**45) 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

**import** java.util.Scanner;

**public** **class** Main {

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

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

String s=sc.nextLine();

System.*out*.println(UserMainCode.*getFormatedString*(s));

}

}

**import** java.util.StringTokenizer;

**public** **class** UserMainCode {

**public** **static** String getFormatedString(String s)

{

StringTokenizer st=**new** StringTokenizer(s," ");

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

**while**(st.hasMoreTokens())

{

String a=st.nextToken();

String b=st.nextToken();

String c=st.nextToken();

sb.append(c.substring(0));

sb.append(" ");

sb.append(b.substring(0,1));

sb.append(".");

sb.append(a.substring(0,1));

//String ss=sb.toString();

}

**return** sb.toString();

}

**46) 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)**

import java.util.Scanner;

public class kape {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int s1=sc.nextInt();

String[] s2 = new String[s1];

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

s2[i] = sc.next();

}

System.out.println(kape1.getSum(s2));

}}

public class kape1 {

public static int getSum(String[] s1) {

int sum=0;

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

for(int j=0;j<s1[i].length();j++){

char c=s1[i].charAt(j);

if(Character.isDigit(c)){

if(c%2!=0)

{

String t=String.valueOf(c);

int n=Integer.parseInt(t);

sum=sum+n; } }}

return sum;

}

}

**47) 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 useer{

public static void main(String[]args)

{

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int []a=new int[100];

int i=0,count=0;

while(n!=0)

{

int num=n%10;

a[i]=num;

i++;

n=n/10;

}

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

{

for(int k=j+1;k<=i-1;k++)

{

if(a[j]==a[k]){

count++;

}

}}

if(count>0)

{

System.out.println("Invalid/not unique");

}

else

{

System.out.println("valid/unique");

}

}}

**48) 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

import java.util.HashMap;

import java.util.Scanner;

public class Main {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

for (int i = 0; i < n; i++) {

hm.put(sc.nextInt(), sc.nextInt());

}

System.out.println(Usermain.getLowest(hm));

}

}

**import** java.lang.reflect.Array;

**import** java.math.BigDecimal;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Set;

**public** **class** Usermain {

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

Collection<Integer> a = hm.values();

Set<Integer> l = hm.keySet();

Object[] m0 = a.toArray();

Arrays.*sort*(m0);

**int** sum = 0;

**for** (**int** i = 0; i < 3; i++) {

sum = sum + (**int**) m0[i];

}

//sum = sum / 3;

**return** sum;

}

}

**49) 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

**import** java.util.Scanner;

**class** Main

{

**public** **static** **void** main(String[] a)

{

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

String s=sc.nextLine();

**if**(s.matches("(#)[A-Z0-9]{6}"))

{

System.*out*.println("valid");

}

**else**

System.*out*.println("invalid");

}

}

**50) 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

import java.util.\*;

public class useerm {

public static String lengthiestString(String s1,int n){

StringBuffer sb=new StringBuffer();

sb.append(s1);

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

{

sb.append(s1.substring(s1.length()-n,s1.length()));

// sb.append(s1.substring(s1.length()-n))

}

return sb.toString();

}

public static void main(String[] args) {

Scanner s=new Scanner(System.in);

System.out.println("enter the String:");

String s1=s.nextLine();

int n=s.nextInt();

System.out.println("the lengthiest string is:"+lengthiestString(s1,n));

}

}

**51) 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

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws ParseException {

Scanner sc=new Scanner(System.in);

String s1=sc.nextLine();

System.out.println(UserMainCode.calculateBornDay(s1));

}

}

import java.text.SimpleDateFormat;

import java.text.ParseException;

import java.util.Date;

public class UserMainCode {

public static String calculateBornDay(String s1) throws ParseException

{

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

SimpleDateFormat sdf1=new SimpleDateFormat("EEEEE");

Date d=sdf.parse(s1);

String s=sdf1.format(d);

return s.toUpperCase();

}

}

**52) 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

**53) 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

import java.util.Date;

import java.text.SimpleDateFormat;

public class Usermaincode

{public static boolean display(String s,String s1,int n)

{

boolean b=false;

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

try{

Date d=sdf.parse(s);

Date d1=sdf.parse(s1);

int y=d.getYear();

int y1=d1.getYear();

int m=d.getMonth();

int m1=d1.getMonth();

int day=d.getDay();

int day1=d1.getDay();

int age=y1-y;

if(m>m1)

age--;

else if(m==m1)

{if(day<day1)

age--;

}

if(age==n)

b=true;

else

b=false;

}

catch(Exception e)

{e.printStackTrace();

}

return b;

}

}

**54) 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** User {

**public** **static** String repeatString (String s)

{

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

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

{

/\* **char** c=s.charAt(i);

**if**(!Character.*isAlphabetic*(c)) \*/

// **if**(!Character.*isAlphabetic*(s.charAt(i))  
&& (!charachter.isWhiteSpace(s.CharAt(i)))

**if**( (!Character.*isAlphabetic*(s.charAt(i)))

&& (!Character.*isWhitespace*(s.charAt(i))) )

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

}

**return** sb.toString();

}

}

**55) 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** User {

**public** **static** String repeatString (String s,**int** n)

{

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

**if**(n>=3)

{

**for**(**int** i=0;i<n;i++)

{

sb.append(s.substring(0,3)).append(" ");

}

}

**else** **if**(n==2)

{

**for**(**int** i=0;i<n;i++)

sb.append(s.substring(0,2)).append(" ");

}

**else** **if**(n==1)

{

**for**(**int** i=0;i<n;i++)

sb.append(s.substring(0,1)).append(" ");

}

**return** sb.toString();

}

}

**56) 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** User {

**public** **static** **float** averageElements(**int** a[],**int** n)

{

**int** c=0,sum=0,k=0;

**float** avg=0;

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

{

c=0;

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

{

**if**(i%j==0)

c++;

}

**if**(c==1)

{

k++;

sum=sum+a[i];

}

}

avg=(**float**)sum/k;

**return** avg;

}

}

**57) 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** User {

**public** **static** **int** getMiddleElement (**int** a[],**int** b[],**int** n)

{

**int** sum=0;

**for**(**int** i=0;i<n;i++)

{

**for**(**int** j=0;j<n;j++)

{

**if**(a[i]==b[j])

sum=sum+a[i];

}

}

**return** sum;

}

}

**58) 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** User {

**public** **static** **int** getMiddleElement (**int** a[])

{

**int** n=a.length;

**int** mid=n/2;

**return** a[mid];

}

}

**59) 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

1. keep the first char if it is 'j'
2. 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** User {

**public** **static** String getString(String s)

{

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

**char** a=s.charAt(0);

**char** b=s.charAt(1);

**if**(a!='j'&& b!='b')

sb.append(s.substring(2));

**else** **if**(a=='j' && b!='b')

sb.append("j").append(s.substring(2));

**else** **if**(a!='j' && b=='b')

sb.append(s.substring(1));

**else**

sb.append(s.substring(0));

**return** sb.toString();

}

}

**60) 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 3:**

Invalid

**public** **class** User {

**public** **static** **int** getValidDate(String s) **throws** ParseException

{

**int** res=0;

// String s1=**null**,s2=**null**,s3=**null**;

**if**(s.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(s);

res=1;

}

**catch** (ParseException e)

{

res=-1;

}

}

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

{

SimpleDateFormat sdf3=**new** SimpleDateFormat("dd-MM-yyyy");

sdf3.setLenient(**false**);

**try**

{

Date d1=sdf3.parse(s);

res=1;

}

**catch** (ParseException e)

{

res=-1;

}

}

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

{

SimpleDateFormat sdf3=**new** SimpleDateFormat("dd/MM/yyyy");

sdf3.setLenient(**false**);

**try**

{

Date d1=sdf3.parse(s);

res=1;

}

**catch** (ParseException e)

{

res=-1;

}

}

**else**

res=0;

**return** res;

}

}

**61) 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** User {

**public** **static** **float** getBoundaryAverage(**int** a[],**int** n)

{

**int** sum=0;

**float** avg=0;

Arrays.*sort*(a);

sum=a[0]+a[n-1];

avg=(**float**)sum/2;

**return** avg;

}

}

**62) 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** User {

**public** **static** **int** countVowels(String s) **throws** ParseException

{

**int** count=0;

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

{

**if**(s.charAt(i)=='a' ||s.charAt(i)=='A' ||

s.charAt(i)=='e' ||s.charAt(i)=='E' ||

s.charAt(i)=='i' ||s.charAt(i)=='I' ||

s.charAt(i)=='o' ||s.charAt(i)=='O' ||

s.charAt(i)=='u' ||s.charAt(i)=='U')

count++;

}

**return** count;

}

}

**63) 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** User {

**public** **static** String getMonthName(String s) **throws** ParseException

{

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

Date d=sdf.parse(s);

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

String month=sdf1.format(d);

**return** month.toUpperCase();

}

}

**64) 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 “**public** **class** User {

” 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

**import** java.util.Scanner;

**public** **class** Main {

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

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

String s=sc.nextLine();

**int** n1=sc.nextInt();

**int** n2=sc.nextInt();

System.*out*.println(UserMainCode.*reverseSubstring*(s,n1,n2));

}

}

**public** **class** UserMainCode {

**public** **static** String reverseSubstring(String s,**int** n1,**int** n2)

{

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

sb.reverse();

String ss=sb.substring(n1,n1+n2);

**return** ss.toString();

}

}

**65) 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** User {

**public** **static** **int** stringFinder(String str,String s1,String s2)

{

**int** res=0;

**if**(str.contains(s1)&&str.contains(s2))

{

**if**(str.indexOf(s1)<str.indexOf(s2))

res=1;

**else**

res=0;

}

**return** res;

}

}

**66) 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** User {

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

{

**int** res=0;

**if**(s.matches("[0-9]{3}(-)[0-9]{3}(-)[0-9]{4}"))

res=1;

**else**

res=-1;

**return** res;

}

}

**68) 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** User {

**public** **static** **int** getNumberOfDays(**int** year,**int** month)

{

GregorianCalendar gc=**new** GregorianCalendar(year,month,1);

**int** days=gc.getActualMaximum(Calendar.*DAY\_OF\_MONTH*);

**return** days;

}

}

**69) 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** User {

**public** **static** String validateNumber(String s)

{

StringTokenizer st=**new** StringTokenizer(s," ");

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

**while**(st.hasMoreTokens())

{

String r=st.nextToken();

**if**(r.equals("is"))

{

sb.append(r.replace("is", "is not"));

}

**else**

sb.append(r);

sb.append(" ");

}

// sb.deleteCharAt((sb.length()-1));

**return** sb.toString();

}

}

**70) 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** User {

**public** **static** String validateNumber(String s)

{

String res=**null**;

**int** count=0;

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

{

**char** c=s.charAt(i);

**if**(Character.*isDigit*(c))

count++;

}

**if**(count==s.length()-1)

{

res=String.*valueOf*(Math.*abs*(Integer.*parseInt*(s)));

}

**else**

res="-1";

**return** res;

}

}

**71) 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** User {

**public** **static** **int** countSeven (**int** n)

{

**int** count=0,r=0;

**while**(n>0)

{

r=n%10;

**if**(r==7)

count++;

n=n/10;

}

**return** count;

}

}

**72) String Processing - III**

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** User {

**public** **static** String getStringUsingNthCharacter (String s)

{

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

StringBuffer sb1=**new** StringBuffer();

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

{

**if**(s.charAt(i)=='x')

{

Sb1.append(s.charAt(i));

}

**else**

{

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

}

}

sb.append(sb1);

**return** sb.toString();

}

}

**73) String Processing - IV**

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** User {

**public** **static** String getStringUsingNthCharacter (String s,**int** n)

{

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

sb.append(s.charAt(0));

**for**(**int** i=1;i<s.length();i=i+n)

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

**return** sb.toString();

}

}

**74) 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** User {

**public** **static** **boolean** compareLastDigit (**int** a,**int** b)

{

**boolean** b1=**false**;

**int** r1=a%10;

**int** r2=b%10;

**if**(r1==r2)

b1=**true**;

**return** b1;

}

}

**75) 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** User {

**public** **static** **int** getDistinctSum (**int** a,**int** b,**int** c)

{

**int** sum=0;

**if**(a==b&&a==c&&b==c)

sum=0;

**else** **if**(a!=b&&b!=c&a==c)

sum=b;

**else** **if**(a==b&&b!=c&&a!=c)

sum=c;

**else** **if**(a!=b&&b!=c&&a!=c)

sum=a+b+c;

**return** sum;

}

}

**int** sum=0;

**if**(a==b&&a==c&&b==c)

sum=0;

**else** **if**(a!=b&&a!=c&b==c)

sum=a;

**else** **if**(a!=b&&b!=c&a==c)

sum=b;

**else** **if**(a==b&&b!=c&&a!=c)

sum=c;

**else**

sum=a+b+c;

**return** sum;

**public** **class** Main {

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

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

**int** a=sc.nextInt();

**int** b=sc.nextInt();

**int** c=sc.nextInt();

**int** sum=0;

**if**(a!=b&&a!=c&&b!=c)

{

sum=a+b+c;

}

**else** **if**(a==c)

{

sum=b;

}

**else** **if**(a==b)

{

sum=c;

}

**else** **if**(b==c)

{

sum=a;

}

**else**

sum=0;

System.*out*.println(sum);

}

}

**76) 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** User {

**public** **static** **boolean** validateString (String s)

{

**boolean** b=**false**;

**if**(s.charAt(1)=='i'&&s.charAt(2)=='x')

b=**true**;

**return** b;

}

}

**77) String Processing**

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** **class** HelloWorld {

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

// String s1="hello world";

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

String s1=sc.nextLine();

String ss=Hello.*display*(s1);

System.*out*.println(ss);

}

}

**public** **class** Hello {

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

StringTokenizer st=**new** StringTokenizer(s1," ");

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

**while**(st.hasMoreTokens())

{

String a=st.nextToken();

String b=st.nextToken();

sb.append(b.substring(b.length()-1));

sb.append(a.substring(1));

sb.append(" ");

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

sb.append(a.substring(0,1));

}

**return** sb.toString();

}

}

**public** **class** WhiteSpaxc {

**public** **static** String validateNumber(String s)

{

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

sb.append(s.substring(s.length()-1));

sb.append(s.substring(1,s.length()-1));

sb.append(s.substring(0,1));

**return** sb.toString();

}

}

**78) Regular Expression - II**

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

**public** **class** Main {

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

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

String s=sc.next();

**boolean** b=User.*validateString*(s);

System.*out*.println(b);

}

}

**public** **class** User {

**public** **static** **boolean** validateString (String s)

{

**boolean** b=**false**;

**if**(s.length()==3)

{

**if**(s.matches("[A-za-z]{3}"))

b=**true**;

}

**return** b;

}

}

**79) 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** User {

**public** **static** String repeatString(String s,**int** n)

{

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

**for**(**int** i=0;i<n;i++)

{

sb.append(s);

}

**return** sb.toString();

}

}

**80) 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** User {

**public** **static** **int** SumOdd (**int** n)

{

**int** sum=0;

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

{

**if**(i%2!=0)

sum=sum+i;

}

**return** sum;

}

}

**81) String Processing - V**

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** Main {

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

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

**int** n=sc.nextInt();

String[] s=**new** String[n];

**for**(**int** i=0;i<n;i++)

s[i]=sc.next();

System.*out*.println(User.*concatString*(s));

}

}

**public** **class** User {

**public** **static** String concatString (String s[])

{

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

sb.append(s[0]);

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

{

sb.append(",");

sb.append(s[i]);

}

**return** sb.toString();

}

}

**82) 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** Main {

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

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

**int** a=sc.nextInt();

**int** b=sc.nextInt();

**int** c=sc.nextInt();

System.*out*.println(User.*calculateUnique*(a,b,c));

}

}

**public** **class** User {

**public** **static** **int** calculateUnique(**int** a,**int** b,**int** c)

{

**int** count=0;

**int**[] s={a,b,c};

**int**[] res=**new** **int**[3];

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

{

res[i]=Math.*abs*(s[i]);

}

count=0;

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

{

**if**(res[i]==res[i+1])

{

count++;

}

}

**return** count+1;

}

}

**public** **class** Main {

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

**int** ct1=0;

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

**int** a=sc.nextInt();

**int** b=sc.nextInt();

**int** c=sc.nextInt();

**if**(a!=b)

ct1=ct1+1;

**if**(a!=c)

ct1=ct1+1;

**if**(b!=c)

ct1=ct1+1;

**if**((a==b) & (b==c))

System.*out*.println("output "+(ct1+1));

**else**

System.*out*.println("output "+ct1);

}

}

**public** **class** Main {

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

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

**int** a=sc.nextInt();

**int** b=sc.nextInt();

**int** c=sc.nextInt();

**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;

}

System.*out*.println(d);

}

}

**83) 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** Main {

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

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

**int** a=sc.nextInt();

**int** b=sc.nextInt();

**char** op=sc.next().charAt(0);

System.*out*.println(User.*calculateUnique*(a,b,op));

}

}

**public** **class** User {

**public** **static** **int** calculateUnique(**int** a,**int** b,**char** op)

{

**int** res=0;

**switch**(op){

**case** '+':

res=a+b;

**case** '-':

res=Math.*abs*(a-b);

**case** '\*':

res=a\*b;

**case** '/':

res=Math.*round*(a/b);

**case** '%':

res=Math.*round*(a%b);

}

**return** res;

}

}

**84) 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** User {

**public** **static** **boolean** scanArray(**int** s[])

{

**boolean** b=**false**;

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

{

**if**(s[i]==100&&s[i+1]==100)

{

b=**true**;

**break**;

}

**else**

b=**false**;

}

**return** b;

}

}

**85) 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** Main {

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

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

**int** n=sc.nextInt();

**int**[] s= **new** **int**[n];

**for**(**int** i=0;i<n;i++)

s[i]=sc.nextInt();

**boolean** b=User.*scanArray* (s);

System.*out*.println(b);

}

}

**public** **class** User {

**public** **static** **boolean** scanArray(**int** s[])

{

**boolean** b=**false**;

**if**(s.length>4)

{

**for**(**int** i=0;i<4;i++)

{

**if**(s[i]==9)

b=**true**;

**else**

b=**false**;

}

}

**else**

{

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

{

**if**(s[i]==9)

b=**true**;

**else**

b=**false**;

}

}

**return** b;

}

}

**86) 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** Main {

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

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

**int** n=sc.nextInt();

String[] s= **new** String[n];

**for**(**int** i=0;i<n;i++)

s[i]=sc.next();

**int** n1=sc.nextInt();

System.*out*.println(User.*countWord* (s,n1));

}

}

**public** **class** User {

**public** **static** **int** countWord (String s[],**int** n1)

{

**int** count=0;

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

{

**if**(s[i].length()==n1)

count++;

}

**return** count;

}

}

**87) 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** Main {

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

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

**int** x1=sc.nextInt();

**int** y1=sc.nextInt();

**int** x2=sc.nextInt();

**int** y2=sc.nextInt();

System.*out*.println(User.*findDistance*(x1,y1,x2,y2));

}

}

**public** **class** User {

**public** **static** **int** findDistance(**int** x1,**int** y1,**int** x2,**int** y2)

{

**double** d=((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2));

**int** res=(**int**)Math.*ceil*(Math.*sqrt*(d));

**return** res;

}

}

**88) 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** Main {

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

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

String s=sc.nextLine();

System.*out*.println(User.*countWord*(s));

}

}

**public** **class** User {

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

{

StringTokenizer st=**new** StringTokenizer(s," ");

**int** count =st.countTokens();

**return** count;

}

}

**89) 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** Main {

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

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

**int** a=sc.nextInt();

**int** b=sc.nextInt();

**int** c=sc.nextInt();

System.*out*.println(User.*getSumMaxMin* (a,b,c));

}

}

**public** **class** User {

**public** **static** **int** getSumMaxMin (**int** a,**int** b,**int** c)

{

**int** sum=0;

**int**[] s={a,b,c};

Arrays.*sort*(s);

sum=s[0]+s[2];

**return** sum;

}

}

**90) 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** Main {

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

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

**int** n=sc.nextInt();

System.*out*.println(User.convertDecimalToBinary(n));

}

}

**public** **class** User {

**public** **static** **long** convertDecimalToBinary(**int** n)

{

String x= Integer.*toBinaryString*(n);

**long** res= Integer.*parseInt*(x);

**long** res=Integer.*parseInt*(Integer.*toBinaryString*(n));

**return** res;

}

}

**91) String Processing - V**

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** Main {

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

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

String s1=sc.next();

**int** n=sc.nextInt();

System.*out*.println(User.*returnLastRepeatedCharacters* (s1,n));

}

}

**public** **class** User {

**public** **static** String returnLastRepeatedCharacters (String s1,**int** n)

{

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

**for**(**int** i=0;i<n;i++)

sb.append(s1.substring(s1.length()-n));

**return** sb.toString();

}

}

**92) Regular Expression - III**

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** Main {

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

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

String s1=sc.next();

**boolean** b=User.*validateString*(s1);

System.*out*.println(b);

}

}

**public** **class** User {

**public** **static** **boolean** validateString(String s1)

{

**boolean** b= **false**;

**if**(!Character.*isDigit*(s1.charAt(0)))

b= **true**;

**else**

b= **false**;

**return** b;

}

}

**93) 3String 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** Main {

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

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

String s1=sc.next();

System.*out*.println(User.*fetchUserName* (s1));

}

}

**public** **class** User {

**public** **static** String fetchUserName (String s1)

{

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

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

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

**return** sb.toString();

}

}

**94) 2 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** Main {

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

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

String s1=sc.next();

System.*out*.println(User.*fetchUserName* (s1));

}

}

**public** **class** User {

**public** **static** String fetchUserName (String s1)

{

**boolean** b=**false**;

StringTokenizer st=**new** StringTokenizer(s1,"@");

String name=st.nextToken();

**return** name;

}

}

**95) 1 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** Main {

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

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

String s1=sc.next();

String s2=sc.next();

**int** n=sc.nextInt();

**boolean** b=User.*isEqual*(s1,s2,n);

System.*out*.println(b);

}

}

**public** **class** User {

**public** **static** **boolean** isEqual(String s1,String s2, **int** n)

{

**boolean** b=**false**;

String i= s1.toLowerCase();

String j= s2.toLowerCase();

**if**(i.charAt(n-1)==j.charAt(n))

b=**true**;

**else**

b=**false**;

**return** b;

}

}

**96) 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 Main {

public static int getDiffArray(int[] n1){

int n2,n3=0,n4=0,i;

for(i=0;i<n1.length-1;i++){

n2=Math.abs(n1[i]-n1[i+1]);

if(n2>n3){

n3=n2;

n4=i+1; }}

return n4;

}

public static void main(String[] args) {

int[] n1={2,4,5,1,9,3,8};

System.out.println(getDiffArray(n1));

}

}