**1. Unique Even Sum**

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

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

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

**Input and Output Format:**

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

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

Refer sample output for formatting specifications.

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

**Sample Input 1:**

4

2

5

1

4

**Sample Output 1:**

6

**Sample Input 2:**

3

1

1

1

**Sample Output 2:**

no even numbers

**Solutions:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

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

**int**[] a = **newint**[20];

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

a[i] = sc.nextInt();

**int** res = User.*addUniqueEven*(a);

**if** (res == -1)

System.*out*.println("no even numbers");

**else**

System.*out*.println(res);

}

}

**publicclass** User {

**publicstaticint** addUniqueEven(**int** a[]) {

**int** i = 0, j = 0, count = 0, sum = 0;

**int** n = a.length;

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

count = 0;

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

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

count++;

}

**if** (count == 0) {

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

sum = sum + a[i];

}

}

**if** (sum == 0)

**return** -1;

**else**

**return**sum;

}

}

**2. Palindrome & Vowels**

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

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

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

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

**Input and Output Format:**

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

Output consists of a single Integer.

Refer sample output for formatting specifications.

**Sample Input 1:**

abceecba

**Sample Output 1:**

valid

**Sample Input 2:**

abcd

**Sample Output 2:**

Invalid

**Solution :**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

String s = sc.next();

**int** res = User.*checkPalindrome*(s);

**if** (res == 1)

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

**else**

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

}

}

**publicclass** User {

**publicstaticint** checkPalindrome(String s) {

**int** res = 0, i = 0, j = 0, count = 0, k = 0;

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

sb.reverse();

**if** (sb.toString().equals(s)) {

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

count = 0;

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

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

count++;

}

**if** (count == 0)

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

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

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

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

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

k++;

}

}

**if** (k >= 2)

res = 1;

**else**

res = 0;

**return** res;

}

}

**3. Strings – Unique & Existing Characters**

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

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

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

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

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

abcxyz

axdef

**Sample Output 1:**

a++ x++

**Sample Input 2:**

ABCDEF

feCBAd

**Sample Output 2:**

ABCDEF

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

String s1 = sc.nextLine();

String s2 = sc.nextLine();

System.*out*.println(User.*replacePlus*(s1, s2));

}

}

**publicclass** User {

**publicstatic** String replacePlus(String s1, String s2) {

String ss1 = s1.toLowerCase();

String ss2 = s2.toLowerCase();

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

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

**char** c = ss1.charAt(i);

**if** (ss2.indexOf(c) == -1)

sb.append('+');

**else**

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

}

**return** sb.toString();

}

}

**4. Longest Word**

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

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

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Welcome to the world of Programming

**Sample Output 1:**

Programming

**Sample Input 2:**

ABC DEF

**Sample Output 2:**

ABC

**Solution:**

**import** java.util.Scanner;

**public** **class** PalindromeMain {

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

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

String s1 = sc.nextLine();

System.*out*.println(Palindrome.*checkPalindrome*(s1));

}

}

**import** java.util.StringTokenizer;

**public** **class** Palindrome {

**public** **static** String checkPalindrome(String s1)

{

**int** res,max=0;

String s2=**null**;

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

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

{

String s=st.nextToken();

res=s.length();

**if**(res>max)

{

max=res;

s2=s;

}

}

**return** s2;

}

}

**5. String Occurences**

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

Return the count as output. Note - Consider case.

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

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

abc bcd abc bcd abc abc

av abc

**Sample Output 1:**

4

**Sample Input 2:**

ABC xyz AAA

w abc

**Sample Output 2:**

0

**Solution:**

**import** java.util.Scanner;

**public** **class** PalindromeMain {

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

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

String s1 = sc.nextLine();

String s2 = sc.nextLine();

System.*out*.println(Palindrome.*checkPalindrome*(s1,s2));

}

}

**import** java.util.StringTokenizer;

**public** **class** Palindrome {

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

{

**int** count=0;

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

StringTokenizer st1=**new** StringTokenizer(s2," ");

String a2=st1.nextToken();

String b2=st1.nextToken();

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

{

String s=st.nextToken();

**if**(s.equalsIgnoreCase(b2))

{

count++;

}

}

**return** count;

}

}

**6. ArrayList Manipulation**

Write a program that performs the following actions:

1. Read 2n integers as input.
2. Create two arraylists to store n elements in each arraylist.
3. Write a function **generateOddEvenList**which accepts these two arraylist as input.
4. The function fetch the odd index elements from first array list and even index elements from second array list and add them to a new array list according to their index.
5. Return the arraylist.

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

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

Note:

- The index of first element is 0.

- Consider 0 as an even number.

- Maintain order in the output array list

**Input and Output Format:**

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

Output consists of a modified arraylist as per step 4.

Refer sample output for formatting specifications.

**Sample Input 1:**

5

12

13

14

15

16

2

3

4

5

6

**Sample Output 1:**

2

13

4

15

6

**Solution :**

**import** java.util.ArrayList;

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

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

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

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

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

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

al1.add(s.nextInt());

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

al2.add(s.nextInt());

a = User.*generateOddEvenList*(al1, al2);

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

System.*out*.println(a.get(i));

}

}

**import** java.util.ArrayList;

**publicclass** User {

**publicstatic** ArrayList<Integer> generateOddEvenList(ArrayList<Integer> a1,

ArrayList<Integer> a2)

{

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

**int** i = 0;

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

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

a.add(a2.get(i));

**else**

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

}

**return** a;

}

}

**7. Duplicate Characters**

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

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

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

hi this is sample test

**Sample Output 1:**

hi tsample

**Sample Input 2:**

ABC DEF

**Sample Output 2:**

ABC DEF

**Solution:**

**import java.util.Scanner;**

**class Main**

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

**{**

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

**String s=sc.nextLine();**

**System.out.println(MainClass.removeDuplicate(s));**

**}}**

**import java.util.HashSet;**

**import java.util.LinkedHashSet;**

**public class MainClass {**

**public static String removeDuplicate(String s)**

**{**

**LinkedHashSet<Character> has=new LinkedHashSet<Character>();**

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

**{**

**has.add(s.charAt(i));**

**}**

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

**for(Character c:has)**

**{**

**sb.append(c);**

**}**

**return sb.toString();**

**}**

**}**

**8. Mastering Hashmap**

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

Your trainer / teacher has given you the following exercise:

1. Read 2n numbers as input where the first number represents a key and second one as value. Both the numbers are of type integers.
2. Write a function **getAverageOfOdd**to find out average of all values whose keys are represented by odd numbers. Assume the average is an int and never a decimal number. Return the average as output. Include this function in class UserMainCode.

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

**Input and Output Format:**

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

Output consists of an integer representing the average.

Refer sample output for formatting specifications.

**Sample Input 1:**

4

2

34

1

4

5

12

4

22

**Sample Output 1:**

8

**Solution:**

**import** java.util.HashMap;

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

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

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

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

hm1.put(s.nextInt(), s.nextInt());

System.*out*.println(User.*getAverageOfOdd*(hm1));

}

}

**import** java.util.HashMap;

**import** java.util.Iterator;

**publicclass** User {

**publicstaticint** getAverageOfOdd(HashMap<Integer, Integer> hm1) {

**int** sum = 0, count = 0;

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

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

**int** key = itr.next();

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

count++;

**int** val = hm1.get(key);

sum = sum + val;

}

}

**int** avg = sum / count;

**return** avg;

}

}

9. **Managers & Hashmaps**

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

1. Read Employee details from the User. The details would include id, designation and salary in the given order. The datatype for id is integer, designation is string and salary is integer.
2. You decide to build two hashmaps. The first hashmap contains employee id as key and designation as value, and the second hashmap contains same employee ids as key and salary as value.
3. The company decides to hike the salary of managers by 5000. You decide to write a function **increaseSalaries** which takes the above hashmaps as input and returns a hashmap with only managers id and their increased salary as output. Include this function in class UserMainCode.

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

2

2

programmer

3000

8

manager

50000

**Sample Output 1:**

8

55000

**Solution :**

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Scanner;

**class** Main {

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

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

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

HashMap<Integer, String> h1 = **new** HashMap<Integer, String>();

HashMap<Integer, Integer> h2 = **new** HashMap<Integer, Integer>();

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

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

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

h1.put(id, sc.next());

h2.put(id, sc.nextInt());

}

hm = User.*dis*(n, h1, h2);

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

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

**int** id = itr.next();

**int** sal = hm.get(id);

System.*out*.println(id);

System.*out*.println(sal);

}

}

}

**import** java.util.HashMap;

**import** java.util.Iterator;

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

**public** **static** HashMap<Integer, Integer> dis(**int** n,

HashMap<Integer, String> h1, HashMap<Integer, Integer> h2) {

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

Iterator<Integer> itr = h1.keySet().iterator();

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

**int** id = itr.next();

String deg = h1.get(id);

**if** (deg.equalsIgnoreCase("manager")) {

hm.put(id, h2.get(id) + 5000);

}

}

**return** hm;

}

}

**10. Check first and last word**

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

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

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

**Input and Output Format:**

Input consists of a string.

Output is an integer.

**Sample Input 1:**

how are you you are how

**Sample Output 1:**

3

**Sample Input 2:**

how is your child

**Sample Output 2:**

8

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

String ss = s.nextLine();

System.*out*.println(User.check(ss));

}

}

**import** java.util.StringTokenizer;

**publicclass** User {

**publicstaticint** check(String s) {

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

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

String[] s1 = **new** String[n];

**int** i = 0, value = 0;

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

s1[i] = st.nextToken();

i++;

}

**if** (s1[0].equals(s1[i - 1]))

value = s1[0].length();

**else**

value = s1[0].length() + s1[i - 1].length();

**return** value;

}

}

**11. Concatenate Characters**

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

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

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

**Input and Output Format:**

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

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

Output consists of a string.

**Sample Input:**

3

ab

a

abcd

**Sample Output:**

bad

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

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

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

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

str[i] = s.next();

System.*out*.println(User.*concatCharacter*(str));

}

}

**publicclass** User {

**publicstatic** String concatCharacter(String[] s) {

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

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

sb.append(s[i].charAt(s[i].length() - 1));

}

**return** sb.toString();

}

}

**12. Anagram**

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

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

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

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

**Input and Output Format:**

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

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

**Sample Input 1:**

eleven plus two

twelve plus one

**Sample Output 1:**

Anagrams

**Sample Input 2:**

orchestra

carthorse

**Sample Output 2:**

Anagrams

**Sample Input 3:**

cognizant

technologies

**Sample Output 3:**

Not Anagrams

**Solutions:**

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

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

String s1 = sc.nextLine();

String s2 = sc.nextLine();

**boolean** b =Anagrams.*check*(s1, s2);

**if** (b == **true**)

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

**else**

System.*out*.println("Not Anagrams");

}

}

**public** **class** Anagrams

{

**public** **static** **boolean** check(String s1,String s2)

{

**boolean** res=**false**;

ArrayList<Character> a1=**new** ArrayList<Character>();

ArrayList<Character> a2=**new** ArrayList<Character>();

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

{

a1.add(s1.charAt(i));

}

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

{

a2.add(s2.charAt(i));

}

Collections.*sort*(a1);

Collections.*sort*(a2);

**if**((a1.containsAll(a2))|| (a2.containsAll(a1)))

{

res=**true**;

}

**return** res;

}

}

**13. Calculate Meter Reading**

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

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

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

FORMULA: (XXXXX-XXXXX)\*4

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

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

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

**Input and Output Format:**

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

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

**Sample Input:**

CSECE12390

CSECE12400

**Sample Output:**

40

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

String s1 = s.nextLine();

String s2 = s.nextLine();

System.*out*.println(User.*calculateMeterReading*(s1, s2));

}

}

**publicclass** User {

**publicstaticint** calculateMeterReading(String s1, String s2) {

String str1 = s1.substring(s1.length() / 2);

String str2 = s2.substring(s2.length() / 2);

**int** a = Integer.*parseInt*(str1);

**int** b = Integer.*parseInt*(str2);

**int** res = (b - a) \* 4;

**return** res;

}

}

**14. Retirement**

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

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

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

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

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

**Input and Output Format:**

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

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

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

and so on...

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

**Sample Input :**  
4  
C1010

02/11/1987

C2020

15/02/1980

C3030

14/12/1952

T4040

20/02/1950

**Sample Output:**

[C3030, T4040]

**Solution:**

**import** java.text.ParseException;

**import** java.util.HashMap;

**import** java.util.Scanner;

**public** **class** NewClassMain {

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

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

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

**int** i=0;

sc.nextLine();

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

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

{

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

}

System.*out*.println(NewClass.*retirement*(hm));

}

}

**public** **class** NewClass {

**public** **static** ArrayList<String> retirement(HashMap<String,String> hm) **throws** ParseException

{

ArrayList<String> a1=**new** ArrayList<String>();

String s="01/01/2014";

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

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

{

String k=itr.next();

String dob=hm.get(k);

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

Date d1=sdf.parse(dob);

Date d2=sdf.parse(s);

Calendar cal=Calendar.*getInstance*();

cal.setTime(d1);

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

cal.setTime(d2);

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

**int** res=y2-y1;

**if**(res>=60)

{

a1.add(k);

}

}

Collections.*sort*(a1);

**return** a1;

}

}

**15. Kaprekar Number**

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

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

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

Example 1:

Input1:9

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

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

Example 2:

Input1:45

Hint:

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

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

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

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

**Input and Output Format:**

Input consists of an integer.

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

**Sample Input 1:**

9

**Sample Output 1:**

Kaprekar Number

**Sample Input 2:**

45

**Sample Output 2:**

Kaprekar Number

**Sample Input 3:**

4

**Sample Output 3:**

Not A Kaprekar Number

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

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

**int** i = User.*getKaprekarNumber*(n);

**if** (i == 1)

System.*out*.println("Kaprekar Number");

**else**

System.*out*.println("Not Kaprekar Number");

}

}

**public** **class** Palindrome {

**public** **static** **int** removeDuplicate(**int** n)

{

**int** temp = n;

**int** sq = n \* n;

**int** sqr=sq;

**int** res = 0;

String sqs=String.*valueOf*(sq);

**int** len=sqs.length();

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

String n1 = a.substring(len/2);

String n2 = a.substring(0,len/2);

**int** i = Integer.*parseInt*(n1);

**int** j = Integer.*parseInt*(n2);

**if** ((i + j) == temp)

res = 1;

**else**

res = -1;

**return** res;

}

}

**16. Vowels**

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

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

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

**Input and Output Format:**

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

Output consists of a string.

**Sample Input :**

What is your name?

**Sample Output :**

Your

**Solution:**

**import** java.text.ParseException;

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) **throws** ParseException {

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

String s = sc.nextLine();

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

}

}

**import** java.util.StringTokenizer;

**publicclass** User {

**publicstatic** String storeMaxVowelWord(String s) {

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

**int** count = 0, max = 0;

String s2 = **null**;

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

String s1 = st.nextToken();

count = 0;

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

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

|| s1.charAt(i) == 'i' || s1.charAt(i) == 'o'

|| s1.charAt(i) == 'u' || s1.charAt(i) == 'A'

|| s1.charAt(i) == 'E' || s1.charAt(i) == 'I'

|| s1.charAt(i) == 'O' || s1.charAt(i) == 'U')

count++;

}

**if** (count > max) {

max = count;

s2 = s1;

}

}

**return** s2;

}

}

**17. Unique Characters**

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

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

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

**Input and Output Format:**

Input consists of a string.

Output consists of an integer.

**Sample Input 1:**

HOWAREYOU

**Sample Output 1:**

7

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

**Sample Input 2:**

MAMA

**Sample Output2:**

-1

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

String s = sc.next();

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

}

}

**publicclass** User {

**publicstaticint** checkUnique(String s) {

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

**int** len = s.length();

**int** i = 0, j = 0, count;

**for** (i = 0; i < len; i++) {

count = 0;

**for** (j = i + 1; j < len; j++) {

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

sb.deleteCharAt(j);

count++;

j--;

len--;

}

}

**if** (count > 0) {

sb.deleteCharAt(i);

i--;

len--;

}

}

**if**(sb.length()==0)

**return** -1;

**else**

**return** sb.length();

}

}

**18. Average of Primes**

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

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

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

**Input and Output Format:**

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

Output consists of a single Integer.

Refer sample output for formatting specifications.

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

**Sample Input 1:**

4

2

5

2

4

**Sample Output 1:**

3

**Solutions:**

**import** java.text.ParseException;

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) **throws** ParseException {

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

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

**int**[] a = **newint**[20];

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

a[i] = sc.nextInt();

System.*out*.println(User.*addPrimeIndex*(a));

}

}

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

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

**int** count=0,sum=0,temp=0;

**int** avg=0;

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

{

count=0;

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

{

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

count++;

}

**if**(count==1)

{

temp++;

sum=sum+a[i];

}

}

avg=sum/temp;

**return** avg;

}

}

19. **ArrayList and Set Operations**

Write a program that performs the following actions:

1. Read 2n integers as input & a set operator (of type char).
2. Create two arraylists to store n elements in each arraylist.
3. Write a function **performSetOperations**which accepts these two arraylist and the set operator as input.
4. The function would perform the following set operations:.

'+' for SET-UNION

'\*' for SET-INTERSECTION

'-' for SET-DIFFERENCE

Refer to sample inputs for more details.

1. Return the resultant arraylist.

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

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

Note:

- The index of first element is 0.

**Input and Output Format:**

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

Output consists of a modified arraylist as per step 4.

Refer sample output for formatting specifications.

**Sample Input 1:**

3

1

2

3

3

5

7

+

**Sample Output 1:**

1

2

3

5

7

**Sample Input 2:**

4

10

9

8

7

2

4

6

8

**\***

**Sample Output 2:**

8

**Sample Input 3:**

4

5

10

15

20

0

10

12

20

-

**Sample Output 3:**

5

15

**Solution:**

import java.util.ArrayList;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

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

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

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

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

al1.add(sc.nextInt());

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

al2.add(sc.nextInt());

char c = sc.next().charAt(0);

res = User.performSetOperations(al1, al2, c);

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

System.out.println(res.get(i));

}

}

import java.util.ArrayList;

import java.util.LinkedHashSet;

public class User {

public static ArrayList<Integer> performSetOperations(

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

LinkedHashSet<Integer> h = new LinkedHashSet<Integer>();

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

switch (c) {

case '+':

al1.addAll(al2);

h.addAll(al1);

al3.addAll(h);

break;

case '\*':

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

for (int j = 0; j < al2.size(); j++) {

if (al1.get(i) == al2.get(j)) {

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

}

}

}

break;

case '-':

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

for (int j = 0; j < al2.size(); j++) {

if (al1.get(i) == al2.get(j)) {

al1.remove(i);

}

}

}

al3.addAll(al1);

break;

}

return al3;

}

}

import java.util.ArrayList;

import java.util.Iterator;

import java.util.Scanner;

class Main

{

public static void main(String[] arg)

{

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

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

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

{

aa.add(sc.nextInt());

}

ArrayList<Integer> aa2=new ArrayList<Integer>();

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

{

aa2.add(sc.nextInt());

}

char c=sc.next().charAt(0);

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

op=MainClass.setOPeration(n, aa, aa2, c);

Iterator<Integer> itr=op.iterator();

while(itr.hasNext())

{

int a=(Integer)itr.next();

System.out.println(a);

}

}}

import java.util.ArrayList;

public class MainClass {

public static ArrayList<Integer> setOPeration

(int n,ArrayList<Integer>aa,ArrayList<Integer>aa2,char c)

{

ArrayList<Integer> aa3= new ArrayList<Integer>();

if(c=='+')

{

aa.removeAll(aa2);

aa.addAll(aa2);

aa3=aa;

}

if(c=='\*')

{

aa.retainAll(aa2);

aa3=aa;

}

if(c=='-')

{

aa.removeAll(aa2);

aa3=aa;

}

return aa3;

}

}

20. **Largest Span**

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

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

.

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

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

**Input and Output Format:**

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

Output consists of a single Integer.

Refer sample output for formatting specifications.

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

**Sample Input 1:**

5

1

2

1

1

3

**Sample Output 1:**

4

**Sample Input 2:**

7

1

4

2

1

4

1

5

**Sample Output 2:**

6

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

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

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

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

a[i] = sc.nextInt();

System.*out*.println(User.*getLargestSpan*(a));

}

}

**publicclass** User {

**publicstaticint** getLargestSpan(**int**[] a) {

**int** len = a.length;

**int** i = 0, j = 0, e = 0, count = 0;

**for** (i = 0; i < len; i++) {

**for** (j = i + 1; j < len; j++) {

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

e = j;

}

}

**if** (e - i > count)

count = e - i;

}

**return** count + 1;

}

}

**21. max Scorer**

Write a program that performs the following actions:

1. Read n strings as input and stores them as an arraylist. The string consists of student information like name and obtained marks of three subjects. Eg: name-mark1-mark2-mark3 [suresh-70-47-12] The mark would range between 0 – 100 (inclusive).
2. Write a function **highestScorer**which accepts these the arraylist and returns the name of the student who has scored the max marks. Assume the result will have only one student with max mark.

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

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

**Input and Output Format:**

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

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

Refer sample output for formatting specifications.

**Sample Input 1:**

3

sunil-56-88-23

bindul-88-70-10

john-70-49-65

**Sample Output 1:**

john

**Solution:**

**import** java.text.ParseException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

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

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

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

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

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

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

a.add(sc.next());

System.***out***.println(User.*highestScorer*(a));

sc.close();

}

}

**import** java.util.ArrayList;

**import** java.util.StringTokenizer;

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

**public** **static** String highestScorer(ArrayList<String> a) {

String ss=**null**,name=**null**,Name=**null**;

**int** m1=0,m2=0,m3=0,sum=0,max=0;

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

{

ss=a.get(i);

StringTokenizer st=**new** StringTokenizer(ss,"-");

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

{

name=st.nextToken();

m1=Integer.*parseInt*(st.nextToken());

m2=Integer.*parseInt*(st.nextToken());

m3=Integer.*parseInt*(st.nextToken());

sum=m1+m2+m3;

**if**(max<sum)

{

max=sum;

Name=name;

}

}

}

**return** Name;

}

}

**22. Max Vowels**

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

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

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

Appreciation is the best way to motivate

**Sample Output 1:**

Appreciation

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

String s = sc.nextLine();

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

}

}

**import** java.util.StringTokenizer;

**publicclass** User {

**publicstatic** String getWordWithMaximumVowels(String s) {

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

**int** count = 0, max = 0;

String res = **null**;

String f = **null**;

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

res = st.nextToken();

count = 0;

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

**if** (res.charAt(k) == 'a' || res.charAt(k) == 'e'

|| res.charAt(k) == 'i' || res.charAt(k) == 'o'

|| res.charAt(k) == 'u' || res.charAt(k) == 'A'

|| res.charAt(k) == 'E' || res.charAt(k) == 'I'

|| res.charAt(k) == 'O' || res.charAt(k) == 'U')

count++;

**if** (count > max) {

max = count;

f = res;

}

}

}

**return** f;

}

}

**23. All Vowels**

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

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

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

acebisouzz

**Sample Output 1:**

valid

**Sample Input 2:**

alphabet

**Sample Output 2:**

invalid

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

String s = sc.nextLine();

**int** res = User.*testOrderVowels*(s);

**if** (res == 1)

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

**else**

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

}

}

**publicclass** User {

**publicstaticint** testOrderVowels(String s1) {

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

**int** res = 0;

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

**if** (s1.charAt(i) == 'a' || s1.charAt(i) == 'A'

|| s1.charAt(i) == 'e' || s1.charAt(i) == 'E'

|| s1.charAt(i) == 'i' || s1.charAt(i) == 'I'

|| s1.charAt(i) == 'o' || s1.charAt(i) == 'O'

|| s1.charAt(i) == 'u' || s1.charAt(i) == 'U') {

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

}

}

**if** (sb.toString().equals("aeiou"))

res = 1;

**else**

res = 0;

**return** res;

}

}

**24. Adjacent Swaps**

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

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

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

forget

**Sample Output 1:**

ofgrte

**Sample Input 2:**

New York

**Sample Output 2:**

eN woYkr

**import** java.util.Scanner;

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

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

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

String string=sc.nextLine();

System.***out***.println(User.*swapPairs*(string));

sc.close();

}

}

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

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

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

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

{

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

{

sb.append(s.charAt(i+1)).append(s.charAt(i));

}

}

**else**

{

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

{

sb.append(s.charAt(i+1)).append(s.charAt(i));

}

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

}

**return** sb.toString();

}

}

**25. Sum of Digits**

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

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

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

abc12de4

**Sample Output 1:**

7

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

String s = sc.next();

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

}

}

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

**int** sum = 0,n=0;

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

**if**( Character.*isDigit*(s.charAt(i)) )

{

**int** c=Character.*getNumericValue*(s.charAt(i));

sum = sum +c;

}

}

String sum1=String.*valueOf*(sum);

**return** sum1;

**}**

**26. Password**

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

Validation Rule:

Atleast 8 characters

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

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

Atleast 1 alphabet(a,B...)

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

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

**Input and Output Format:**

Input consists of a String.

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

**Sample Input 1:**

cts@1010

**Sample Output 1:**

Valid

**Sample Input 2:**

punitha3

**Sample Output 2:**

Invalid

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

String s = sc.next();

**boolean** flag = User.*validatePassword*(s);

**if** (flag == **true**)

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

**else**

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

}

}

**publicclass** User {

**publicstaticboolean** validatePassword(String s) {

**int** number = 0, c = 0, sp = 0;

**boolean** flag = **false**;

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

**if** (s.length() >= 8) {

**if** (Character.*isDigit*(s.charAt(i))) {

number++;

}

**if** (Character.*isLetter*(s.charAt(i))) {

c++;

} **else** {

**if** (s.charAt(i) != ' '&& !Character.*isDigit*(s.charAt(i))

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

sp++;

}

}

}

**if** (number >= 1 && c >= 1 && sp >= 1)

flag = **true**;

**return** flag;

}

}

 **27. Employee Bonus**

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

1. Read Employee details from the User. The details would include id, DOB (date of birth) and salary in the given order. The datatype for id is integer, DOB is string and salary is integer.
2. You decide to build two hashmaps. The first hashmap contains employee id as key and DOB as value, and the second hashmap contains same employee ids as key and salary as value.
3. If the age of the employee in the range of 25 to 30 years (inclusive), the employee should get bonus of 20% of his salary and in the range of 31 to 60 years (inclusive) should get 30% of his salary. store the result in TreeMap in which Employee ID as key and revised salary as value. Assume the age is caculated based on the date 01-09-2014. (Typecast the bonus to integer).
4. Other Rules:

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

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

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

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

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

**Input and Output Format:**

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

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

2

1010

20-12-1987

10000

2020

01-01-1985

14400

**Sample Output 1:**

1010

12000

2020

17280  
  
 **Solution:**

**import** java.text.ParseException;

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

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

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

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

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

TreeMap<Integer,Integer> t=**new** TreeMap<Integer,Integer>();

HashMap<Integer,String> h1=**new** HashMap<Integer,String>();

HashMap<Integer,Integer> h2=**new** HashMap<Integer,Integer>();

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

{

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

h1.put(id, sc.next());

h2.put(id, sc.nextInt());

}

t=User.*calSalary*(h1,h2);

Iterator<Integer> it1=t.keySet().iterator();

**while**(it1.hasNext())

{

**int** id=it1.next();

**int** val=t.get(id);

System.***out***.println(id);

System.***out***.println(val);

}

sc.close();

}

}

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

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

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

**public** **static** TreeMap<Integer,Integer> calSalary(HashMap<Integer,String> h1, HashMap<Integer,Integer> h2) **throws** ParseException {

TreeMap<Integer,Integer> t=**new** TreeMap<Integer,Integer>();

Iterator<Integer> it1=h1.keySet().iterator();

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

String ss="01-09-2014";

**int** new\_sal=0;

**while**(it1.hasNext())

{

**int** id1=it1.next();

String dob=h1.get(id1);

**int** salary=h2.get(id1);

Date d1=sd.parse(dob);

Date d2=sd.parse(ss);

d1=sd.parse(dob);

**int** y1=d1.~~getYear~~();

**int** y2=d2.~~getYear~~();

**int** year=Math.*abs*(y1-y2);

**if**(year>=25 && year<=30)

{

new\_sal=salary+(salary\*20/100);

t.put(id1,new\_sal);

}

**else** **if**(year>=31 && year<=60)

{

new\_sal=salary+(salary\*30/100);

t.put(id1,new\_sal);

}

**else**

;

}

**return** t;

}

}

 **28. Grade Calculator**

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

1. Read student details from the User. The details would include roll no, mark in the given order. The datatype for id is integer, mark is integer.
2. You decide to build a hashmap. The hashmap contains roll no as key and mark as value.
3. BUSINESS RULE:

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

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

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

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

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

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

**Input and Output Format:**

Input consists of employee details. The first number indicates the size of the students. The next two values indicate the roll id, mark.

Output consists of a single string.

Refer sample output for formatting specifications.

**Sample Input 1:**

2

1010

80

100

40

**Sample Output 1:**

100

FAIL

1010

GOLD

**Solution:**

import java.util.HashMap;

import java.util.Iterator;

import java.util.Scanner;

import java.util.TreeMap;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int i;

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

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

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

}

TreeMap<Integer, String> t = new TreeMap<Integer, String>();

t.putAll(User.display(n, hm));

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

while (it.hasNext()) {

int r = it.next();

String g = t.get(r);

System.out.println(r);

System.out.println(g);

}}}

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.TreeMap;

**publicclass** User {

**publicstatic** TreeMap<Integer, String> display(**int** n,

HashMap<Integer, Integer> h) {

TreeMap<Integer, String> t = **new** TreeMap<Integer, String>();

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

**while** (i.hasNext()) {

**int** r = i.next();

**int** m = h.get(r);

**if** (m >= 80)

t.put(r, "GOLD");

**elseif** (m < 80 && m >= 60)

t.put(r, "SILVER");

**elseif** (m < 60 && m >= 45)

t.put(r, "BRONZE");

**else**

t.put(r, "FAIL");

}

**return** t;

}

}

**29. DigitSum**

Write a program to read a non-negative integer n, compute the sum of its digits. If sum is greater than 9 repeat the process and calculate the sum once again until the final sum comes to single digit.Return the single digit.  
Include a class UserMainCode with a static method **getDigitSum** which accepts the integer value. The return type is integer.  
Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of a integer.  
Output consists of integer.  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
9999  
**Sample Output 1:**  
9  
  
**Sample Input 2:**  
698  
**Sample Output 2:**  
5

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

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

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

}

}

**publicclass** User {

**publicstaticint** getDigitSum(**int** n) {

**int** sum = 0;

**while** (n > 10) {

**int** r = 0;

sum = 0;

**while** (n != 0) {

r = n % 10;

sum = sum + r;

n = n / 10;

}

n = sum;

}

**return** sum;

}

}

**30. Anagrams**

Write a program to read two strings and checks if one is an anagram of the other.  
An anagram is a word or a phrase that can be created by rearranging the letters of another given word or phrase. We ignore white spaces and letter case. All letters of 'Desperation' can be rearranged to the phrase 'A Rope Ends It'.  
Include a class UserMainCode with a static method **checkAnagram** which accepts the two strings. The return type is boolean which is TRUE / FALSE.  
Create a Class Main which would be used to accept the two strings and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of two strings.  
Output consists of TRUE / FALSE.  
Refer sample output for formatting specifications.  
**Sample Input 1:**  
tea  
eat  
**Sample Output 1:**  
TRUE  
  
**Sample Input 2:**  
Desperation  
A Rope Ends It  
**Sample Output 2:**  
TRUE

**Solution:**

**public** **class** Anagrams

{

**public** **static** **boolean** check(String s1,String s2)

{

**boolean** res=**false**;

ArrayList<Character> a1=**new** ArrayList<Character>();

ArrayList<Character> a2=**new** ArrayList<Character>();

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

{

a1.add(s1.charAt(i));

}

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

{

a2.add(s2.charAt(i));

}

Collections.*sort*(a1);

Collections.*sort*(a2);

**if**((a1.containsAll(a2))|| (a2.containsAll(a1)))

{

res=**true**;

}

**return** res;

}

}

**31. Shift Left**

Write a program to read a integer array of scores, and return a version of the given array where all the 5's have been removed. The remaining elements should shift left towards the start of the array as needed,  
  
and the empty spaces at the end of the array should be filled with 0.  
  
So {1, 5, 5, 2} yields {1, 2, 0, 0}.  
  
Include a class UserMainCode with a static method shiftLeft which accepts the integer array. The return type is modified array.  
  
Create a Class Main which would be used to accept the integer array and call the static method present in UserMainCode.  
  
**Input and Output Format:**

Input consists of an integer n which is the number of elements followed by n integer values.  
Output consists of modified array.  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
7  
1  
5  
2  
4  
5  
3  
5  
  
**Sample Output 1:**  
1  
2  
4  
3  
0  
0  
0

**Solution:**

**import** java.util.Scanner;

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

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

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

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

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

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

a[i]=s.nextInt();

**int** res[]=User.*shiftLeft*(a,n);

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

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

}

}

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

**public** **static** **int**[] shiftLeft(**int** a[],**int** n)

{

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

**int** k=0;

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

{

**if**(a[i]!=5)

{

b[k]=a[i];

k++;

}

}

**return** b;

}

}

**32. Word Count**

Given a string array (s) with each element in the array containing alphabets or digits. Write a program to add all the digits in every string and return the sum as an integer. If two digits appear simultaneously do not consider it as one number. Ex- For 'Hyderabad 21' consider 2 and 1 as two digits instead of 21 as a number.  
  
Include a class UserMainCode with a static method **sumOfDigits** which accepts the string array. The return type is the integer formed based on rules.  
Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of a an integer indicating the number of elements in the string array.  
Output consists of a integer .  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
5  
AAA1  
B2B  
4CCC  
A5  
ABCDE  
**Sample Output 1:**  
12  
  
**Sample Input 2:**  
3  
12  
C23  
5CR2  
**Sample Output 2:**  
15

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** 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.*sumOfDigits*(s));

}

}

**publicclass** User {

**publicstaticint** sumOfDigits(String[] ss) {

**int** sum = 0, n = 0;

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

String s = ss[i];

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

**if** (Character.*isDigit*(s.charAt(k))) {

n = Character.*getNumericValue*(s.charAt(k));

sum = sum + n;

}

}

}

**return** sum;

}

}

**33. Prefix finder**

Given a string array (s) with each element in the array containing 0s and 1s. Write a program to get the number of strings in the array where one String is getting as prefixed in other String in that array .  
Example 1: Input: {10,101010,10001,1111} Output =2 (Since 10 is a prefix of 101010 and 10001)  
Example 2: Input: {010,1010,01,0111,10,10} Output =3(01 is a prefix of 010 and 0111. Also, 10 is a prefix of 1010) Note: 10 is NOT a prefix for 10.  
  
Include a class UserMainCode with a static method **findPrefix** which accepts the string array. The return type is the integer formed based on rules.  
Create a Class Main which would be used to accept the string and integer and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of a an integer indicating the number of elements in the string array followed by the array.  
Output consists of a integer .  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
4  
0  
1  
11  
110  
**Sample Output 1:**  
3

**Solution:**

**import** java.util.HashSet;

**import** java.util.Scanner;

**public** **class** Piddi {

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

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

**int** size = sc.nextInt();

String input[] = **new** String[size];

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

input[i] = sc.next();

}

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

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

hs.add(input[i]);

}

size = hs.size();

**int** i = 0;

**int** count = 0;

**for** (i = 0; i < size; i++) {

**for** (**int** j = 1; j < size; j++) {

**if** (input[i].equals(input[j]) == **false**) {

**if** (input[j].startsWith(input[i])) {

count++;

}

}

}

}

System.*out*.println(count);

}

}

**34. Commons**

Given two arrays of strings,return the count of strings which is common in both arrays. Duplicate entries are counted only once.  
Include a class UserMainCode with a static method **countCommonStrings** which accepts the string arrays. The return type is the integer formed based on rules.  
Create a Class Main which would be used to accept the string arrays and integer and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of a an integer indicating the number of elements in the string array followed by the array.  
Output consists of a integer .  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
3  
a  
c  
e  
3  
b  
d  
e  
**Sample Output 1:**  
1  
  
**Sample Input 2:**  
5  
ba  
ba  
black  
sheep  
wool  
5  
ba  
ba  
have  
any  
wool  
**Sample Output 2:**  
2

**Solution:**

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

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

String[] s1 = **new** String[n1];

**for** (**int** i = 0; i < n1; i++) {

s1[i] = sc.next();

}

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

String[] s2 = **new** String[n2];

**for** (**int** i = 0; i < n2; i++) {

s2[i] = sc.next();

}

System.*out*.println(User.*countCommonStrings*(s1, s2, n1, n2));

}

}

**import** java.util.HashSet;

**import** java.util.Iterator;

**public** **class** Palindrome {

**public** **static** **int** removeDuplicate(String[] words1,String[] words2)

{

**int** count=0;

HashSet<String> s1=**new** HashSet<String>();

HashSet<String> s2=**new** HashSet<String>();

**for**(**int** i=0;i<words1.length;i++)

{

s1.add(words1[i]);

}

**for**(**int** i=0;i<words2.length;i++)

{

s2.add(words2[i]);

}

Iterator<String> it1=s1.iterator();

**while**(it1.hasNext())

{

String its1=it1.next();

Iterator<String> it2=s2.iterator();

**while**(it2.hasNext())

{

String its2=it2.next();

**if**(its1.equals(its2))

{

count++;

}

}

}

**return** count;

}

}

**35. Sequence Sum**

Write a program to read a non-negative integer n, and find sum of fibonanci series for n number..  
  
Include a class UserMainCode with a static method **getFibonacciSum** which accepts the integer value. The return type is integer.  
  
The fibonacci seqence is a famous bit of mathematics, and it happens to have a recursive definition.  
  
The first two values in the sequnce are 0 and 1.  
  
Each subsequent value is the sum of the previous two values, so the whole seqence is 0,1,1,2,3,5 and so on.  
  
You will have to find the sum of the numbers of the Fibonaaci series for a given int n.  
  
Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
  
Input consists of a integer.  
  
Output consists of integer.  
  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
  
5  
  
**Sample Output 1:**  
  
7

**Solution:**

**import**java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) {

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

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

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

}

}

**publicclass** User {

**publicstaticint** getFibonacciSum(**int** n) {

**int** a = 0, b = 1, c = 0, d = 1;

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

c = a + b;

a = b;

b = c;

d = d + c;

}

**return** d;

}

}

**36. Email Validation**

Write a program to read a string and validate the given email-id as input.  
Validation Rules:  
1. Ensure that there are atleast 5 characters between '@' and '.'  
2. There should be only one '.' and one '@' symbol.  
3. The '.' should be after the '@' symbol.  
4. There must be atleast three characters before '@'.  
5. The string after '.' should only be 'com'  
  
Include a class UserMainCode with a static method **ValidateEmail** which accepts the string. The return type is TRUE / FALSE as per problem.  
Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of a string.  
Output consists of TRUE / FALSE.  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
test@gmail.com  
**Sample Output 1:**  
TRUE  
  
**Sample Input 2:**  
academy@xyz.com  
**Sample Output 2:**  
FALSE

**Solution:**

**import** java.util.Scanner;

**class** Main {

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

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

String email = sc.next();

System.*out*.println(User.*ValidateEmail*(email));

}

}

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

**public** **static** **boolean** ValidateEmail(String email) {

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

**if** (email.matches("[a-zA-Z0-9]{3,}(@)[a-zA-Z]{5,}(.)(com)"))

b = **true**;

**return** b;

}

}

**37. Symmetric Difference**

Write a program to read two integer array and calculate the symmetric difference of the two arrays. Finally Sort the array.  
Symmetric difference is the difference of A Union B and A Intersection B ie. [ (A U B) - (A ^ B)]  
Union operation merges the two arrays and makes sure that common elements appear only once. Intersection operation includes common elements from both the arrays.  
Ex - A={12,24,7,36,14} and B={11,26,7,14}.  
A U B ={ 7,11,12,14,24,26,36} and  
A ^ B = {7,14}  
Symmetric difference of A and B after sorting= [A U B] - [ A ^ B] = {11,12,24,26,36}.  
Include a class UserMainCode with a static method **getSymmetricDifference** which accepts the integer array. The return type is an integer array.  
Create a Class Main which would be used to accept the two integer arrays and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of an integer n which is the number of elements followed by n integer values. The same sequnce is followed for the next array.  
Output consists of sorted symmetric difference array.  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
5  
11  
5  
14  
26  
3  
3  
5  
3  
1  
**Sample Output 1:**  
1  
11  
14  
26

**Solution:**

**public** **class** Main {

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

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

**int** n1=sc.nextInt();

**int**[] a=**new** **int**[n1];

**for**(**int** i=0;i<n1;i++)

a[i]=sc.nextInt();

**int** n2=sc.nextInt();

**int**[] b= **new** **int**[n2];

**for**(**int** i=0;i<n2;i++)

b[i]=sc.nextInt();

**int**[] res=User.*display*(a,b,n1,n2);

**for**(**int** i=0;i<res.length;i++)

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

}

}

**public** **class** User {

**public** **static** **int**[] display(**int** a[],**int** b[],**int** n1,**int** n2)

{

TreeSet<Integer> ts1=**new** TreeSet<Integer>();

TreeSet<Integer> ts2=**new** TreeSet<Integer>();

TreeSet<Integer> ts3=**new** TreeSet<Integer>();

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

**for**(**int** i=0;i<a.length;i++)

ts1.add(a[i]);

**for**(**int** i=0;i<b.length;i++)

ts2.add(b[i]);

ts1.addAll(ts2);

**for**(**int** i=0;i<n1;i++)

{

**for**(**int** j=0;j<n2;j++)

{

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

ts3.add(a[i]);

}

}

ts1.removeAll(ts3);

aa.addAll(ts1);

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

**for**(**int** i=0;i<res.length;i++)

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

**return** res;

}

}

**38. Day of Week**

Write a program to read a string  containing date in DD/MM/YYYY format and prints the day of the week that date falls on.  
Return the day in lowercase letter (Ex: monday)  
  
Include a class UserMainCode with a static method **getDayOfWeek** which accepts the string. The return type is the string.  
Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of a string.  
Output consists of a string.  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
02/04/1985  
**Sample Output 1:**  
tuesday

**Solution:**

**import** java.text.ParseException;

**import** java.util.Scanner;

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

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

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

String s1=s.next();

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

}

}

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

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

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

**public** **static** String findOldDate(String s1) **throws** ParseException

{

SimpleDateFormat sd1=**new** SimpleDateFormat("dd-MM-yyyy");

Date d1=sd1.parse(s1);

SimpleDateFormat sd2=**new** SimpleDateFormat("EEEEE");

String name=sd2.format(d1);

**return** name.toLowerCase();

}

}

**39. Addtime**

Write a program to read  two String variables containing time intervals in hh:mm:ss format. Add the two time intervals and return a string in days:hours:minutes:seconds format where DD is number of days.  
Hint: Maximum value for hh:mm:ss is 23:59:59  
  
Include a class UserMainCode with a static method **addTime** which accepts the string values. The return type is the string.  
Create a Class Main which would be used to accept the two string values and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of two string.  
Output consists of a string.  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
12:45:30  
13:50:45  
**Sample Output 1:**  
1:2:36:15  
  
**Sample Input 2:**  
23:59:59  
23:59:59  
**Sample Output 2:**  
1:23:59:58

**Solution:**

**import** java.text.ParseException;

**import** java.util.Scanner;

**publicclass** Main {

**publicstaticvoid** main(String[] args) **throws** ParseException {

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

String s1 = sc.next();

String s2 = sc.next();

System.*out*.println(User.*addTime*(s1, s2));

}

}

**import** java.util.StringTokenizer;

**public** **class** Palindrome {

**public** **static** String removeDuplicate(String a,String b)

{

StringTokenizer st1=**new** StringTokenizer(a,":");

StringTokenizer st2=**new** StringTokenizer(b,":");

**int** h1=Integer.*parseInt*(st1.nextToken());

**int** m1=Integer.*parseInt*(st1.nextToken());

**int** s1=Integer.*parseInt*(st1.nextToken());

**int** d=0;

**int** h2=Integer.*parseInt*(st2.nextToken());

**int** m2=Integer.*parseInt*(st2.nextToken());

**int** s2=Integer.*parseInt*(st2.nextToken());

**int** m,h,s;

m=m1+m2;

h=h1+h2;

s=s1+s2;

**while**(s>=60)

{

m=m+1;

s=s-60;

}

**while**(m>=60)

{

h=h+1;

m=m-60;

}

**while**(h>=24)

{

d=d+1;

h=h-24;

}

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

sb.append(d).append(":").append(h).append(":").append(m).append(":").append(s);

**return** sb.toString();

}

}

**40. ISBN Validation**

Write a program to read a string and validate the given ISBN as input.  
Validation Rules:  
1. An ISBN (International Standard Book Number) is a ten digit code which uniquely identifies a book.  
2. To verify an ISBN you calculate 10 times the first digit, plus 9 times the second digit, plus 8 times the third ..all the way until you add 1 times the last digit.  
If the final number leaves no remainder when divided by 11 the code is a valid ISBN.  
Example 1:  
Input:0201103311  
Calculation: 10\*0 + 9\*2 + 8\*0 + 7\*1 + 6\*1 + 5\*0 + 4\*3 + 3\*3 + 2\*1 + 1\*1= 55.  
55 mod 11 = 0  
Hence the input is a valid ISBN number  
Output: true  
Include a class UserMainCode with a static method **validateISBN** which accepts the string. The return type is TRUE / FALSE as per problem.  
Create a Class Main which would be used to accept the string and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of a string.  
Output consists of TRUE / FALSE.  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
0201103311  
**Sample Output 1:**  
TRUE

**Solution:**

**import** java.util.Scanner;

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

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

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

String isbn=s.next();

**boolean** b=User.*validateISBN*(isbn);

System.***out***.println(b);

}

}

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

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

**public** **static** **boolean** validateISBN(String isbn)

{

**int** sum=0,k=10;

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

{

**int** a=Character.*getNumericValue*(isbn.charAt(i));

sum=sum+(a\*k);

k--;

}

**if**(sum%11==0)

**return** **true**;

**else**

**return** **false**;

}

}

**41. Date Format**

Write a program to read  two String variables in DD-MM-YYYY.Compare the two dates and return the older date in 'MM/DD/YYYY' format.  
  
Include a class UserMainCode with a static method **findOldDate** which accepts the string values. The return type is the string.  
Create a Class Main which would be used to accept the two string values and call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of two string.  
Output consists of a string.  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
05-12-1987  
8-11-2010  
**Sample Output 1:**  
12/05/1987

**Solution:**

**import** java.text.ParseException;

**import** java.util.Scanner;

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

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

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

String s1=s.next();

String s2=s.next();

System.***out***.println(User.*findOldDate*(s1,s2));

}

}

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**public** **class** Palindrome {

**public** **static** String removeDuplicate(String s1,String s2) **throws** ParseException

{

SimpleDateFormat sd1=**new** SimpleDateFormat("dd-MM-yyyy");

Date d1=sd1.parse(s1);

Date d2=sd1.parse(s2);

String res=**null**;

SimpleDateFormat sfd2=**new** SimpleDateFormat("MM/dd/yyyy");

**if**(d1.compareTo(d2)<0)

{

res=sfd2.format(d1);

}

**else**

{

res=sfd2.format(d2);

}

**return** res;

}

}

**42. Interest calculation**

1. Read account details from the User. The details would include id, DOB (date of birth) and amount in the given order. The datatype for id is string, DOB is string and amount is integer.  
2. You decide to build two hashmaps. The first hashmap contains employee id as key and DOB as value, and the second hashmap contains same employee ids as key and amount as value.  
3. Rate of interest as on 01/01/2015:  
    a. If the age greater than or equal to 60 then interest rate is 10% of Amount.  
    b.If the age less then to 60 and greater than or equal to 30 then interest rate is 7% of Amount.  
    v. If the age less then to 30 interest rate is 4% of Amount.  
4. Revised Amount= principle Amount + interest rate.  
5.  You decide to write a function **calculateInterestRate** which takes the above hashmaps as input and returns the treemap  as output. Include this function in class UserMainCode.  
  
Create a Class Main which would be used to read employee details in step 1 and build the two hashmaps. Call the static method present in UserMainCode.  
  
**Input and Output Format:**  
Input consists of account details. The first number indicates the size of the acoount. The next three values indicate the user id, DOB and amount. The Employee DOB format is “dd-mm-yyyy”  
Output consists of the user id and the amount for each user one in a line.  
Refer sample output for formatting specifications.  
  
**Sample Input 1:**  
4  
SBI-1010  
20-01-1987  
10000  
SBI-1011  
03-08-1980  
15000  
SBI-1012  
05-11-1975  
20000  
SBI-1013  
02-12-1950  
30000  
**Sample Output 1:**  
SBI-1010:10400  
SBI-1011:16050  
SBI-1012:21400  
SBI-1013:33000

**import** java.text.ParseException;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Scanner;

**import** java.util.Set;

**import** java.util.TreeMap;

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

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

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

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

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

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

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

String s = sc.next();

hm.put(s, sc.next());

hm1.put(s, sc.next());

}

TreeMap<String, String> tr = **new** TreeMap<String, String>();

tr = MainClass.*calculateInterestRate*(hm, hm1);

Set<String> itr4 = tr.keySet();

Iterator<String> it = itr4.iterator();

**while** (it.hasNext()) {

String id = it.next();

String val = tr.get(id);

System.***out***.println(id + " : " + val);

}

}

}

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Collections;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.TreeMap;

**public** **class** MainClass {

**public** **static** TreeMap<String, String> calculateInterestRate(HashMap<String, String> hm, HashMap<String, String> hm1)

**throws** ParseException {

TreeMap<String, String> tr = **new** TreeMap<String, String>();

String D = "01/01/2015";

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

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

String id = itr.next();

String dob = hm.get(id);

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

Date D1 = sdf.parse(dob);

**int** y1 = D1.~~getYear~~();

SimpleDateFormat sdf1 = **new** SimpleDateFormat("dd/mm/yyyy");

Date D2 = sdf1.parse(D);

**int** y0 = D2.~~getYear~~();

**if** (y0 - y1 >= 60) {

**int** i = Integer.*valueOf*(hm1.get(id));

**int** val = (**int**) (i + (0.1 \* i));

String val1 = String.*valueOf*(val);

tr.put(id, val1);

}

**if** (y0 - y1 < 60 && y0 - y1 >= 30) {

**int** i = Integer.*valueOf*(hm1.get(id));

**int** val = (**int**) (i + (0.07 \* i));

String val1 = String.*valueOf*(val);

tr.put(id, val1);

}

**if** (y0 - y1 < 30) {

**int** i = Integer.*valueOf*(hm1.get(id));

**int** val = (**int**) (i + (0.04 \* i));

String val1 = String.*valueOf*(val);

tr.put(id, val1);

}

}

**return** tr;

}

}

**43. Discount rate calculation**

Write a  program to calculate discount of the acccount holders based on the transaction amount and registration date using below mentioned prototype:  
1. Read account details from the User. The details would include id, DOR (date of registration) and transaction amount in the given order. The datatype for id is string, DOR is string and transaction amount is integer.  
2. You decide to build two hashmaps. The first hashmap contains employee id as key and DOR as value, and the second hashmap contains same employee ids as key and amount as value.  
3. Discount Amount as on 01/01/2015:  
    a. If the transaction amount greater than or equal to 20000 and registration greater than or equal to 5     year then discount rate is 20% of transaction amount.  
    b. If the transaction amount greater than or equal to 20000 and registration less then to 5 year then     discount rate is 10% of transaction amount.  
    c. If the transaction amount less than to 20000 and registration greater than or equal to 5 year then     discount rate is 15% of transaction amount.  
    d. If the transaction amount less than to 20000 and registration less then to 5 year then discount rate     is 5% of transaction amount.  
4. You decide to write a function **calculateDiscount** which takes the above hashmaps as input and returns the treemap  as output. Include this function in class UserMainCode.  
  
Create a Class Main which would be used to read employee details in step 1 and build the two hashmaps. Call the static method present in UserMainCode.  
**Input and Output Format:**  
Input consists of transaction details. The first number indicates the size of the employees. The next three values indicate the user id, user DOR and transaction amount. The DOR (Date of Registration) format is “dd-mm-yyyy”  
Output consists of a string which has the user id and discount amount one in a line for each user.  
Refer sample output for formatting specifications.  
**Sample Input 1:**  
4  
A-1010  
20-11-2007  
25000  
B-1011  
04-12-2010  
30000  
C-1012  
11-11-2005  
15000  
D-1013  
02-12-2012  
10000  
**Sample Output 1:**  
A-1010:5000  
B-1011:3000  
C-1012:2250  
D-1013:500

**Solution:**

public class main {

public static void main(String []args){

Scanner sc=new Scanner(System.in);

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

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

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

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

{

String id=sc.nextLine();

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

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

}

TreeMap<String,Integer>tm=new TreeMap<String,Integer>();

tm=Usermaincode.findDiscountRate(hm,hm1);

Iterator<String> it=tm.keySet().iterator();

while(it.hasNext())

{

String n=it.next();

int fac=tm.get(n);

System.out.println(n+":"+fac);

}

}

public class UserMaincode

{

public static TreeMap<String,Integer> findDiscountRate (HashMap<String,String>hm,HashMap<String,Integer>hm1) throws ParseException

{

TreeMap<String,Integer> tm=new TreeMap<String,Integer>();

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

Iterator<String> itr1=hm.keySet().iterator();

while(itr1.hasNext())

{

try

{

String id=itr1.next();

String dor=hm.get(id);

int am=hm1.get(id);

Date d1=sdf.parse(dor);

String s1="01-01-2015";

Date d2=sdf.parse(s1);

int y1=d1.getYear();

int m1=d1.getMonth();

int day1=d1.getDay();

int y2=d2.getYear();

int m2=d2.getMonth();

int day2=d2.getDay();

int exp=Math.abs(y1-y2);

if(m1==m2)

{

if(day2>day1)

exp--;

}

if(m2>m1)

exp--;

if(am>=20000 && exp>=5)

{

int dis=(int) (0.20\*am);

tm.put(id,dis);

}

else if(am>=20000 && exp<5)

{

int dis=(int) (0.1\*am);

tm.put(id,dis);

}

else if(am<20000 && exp>=5)

{

int dis=(int) (0.15\*am);

tm.put(id,dis);

}

else if(am<20000 && exp<5)

{

int dis=(int) (0.05\*am);

tm.put(id,dis);

}

}

catch(Exception e){

System.out.println(e);

}

}

return tm;

}

}

}

}