//Given an array of integers, every element appears twice except for one. Find that single one.

//PseudoCode

/\*

\* Step1: Calculate the first occurrence of integer.

\* Step2: Calculate the last occurrence of integer.

\* Step3: If first occurrence is equal to the last occurrence then the integer is found4

\* step4: Iterate from step 1-3 for all the elements in arrayList.

\*/

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** FindingNonRepeatIntegerInArray {

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

// **TODO** Auto-generated method stub

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

al.add(5);

al.add(5);

al.add(7);

al.add(8);

al.add(8);

al.add(10);

al.add(10);

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

**if**( al.indexOf(al.get(i)) == al.lastIndexOf(al.get(i)) ){

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

}

}

}

}

//StackTrace

/\*

\* i al[i] al.indexOf(al.get(i)) al.lastIndexOf(al.get(i))

\* (first occurrence) (last occurrence)

\*

\* 0 5 0 1

\* 1 5 0 1

\* 2 7 2 2

\* 3 8 3 4

\* 4 8 3 4

\* 5 10 5 6

\* 6 10 5 6

\*/

//TestCases

/\*

\*

\* input: 5, 7, 7, 8, 8, 10

\* output: 5

\*

\* input: 5,5, 7, 8, 8, 10

\* output: 7

\*

\* input:5,5,7, 7, 8, 10

\* output: 8

\*

\*/

/\*

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.  
For example, given n = 3, a solution set is:  
"((()))", "(()())", "(())()", "()(())", "()()()"

\*/

//PseudoCode

/\*

\* Step1: Keep Track of no: of open and close brackets using open and close variables.

\* Step2: If both open and close are equal to zero print string.

\* Step3: If open>0 then add '{' to string,open-1;close+1.

\* Step4: If close>0 then add '{' to string,open-1;close+1.

\* Step5: Repeat step 3,4 till open = close =0.

\*/

**public** **class** GenerateParanthesisPemutations {

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

// **TODO** Auto-generated method stub

*paranthesis*("",2,0);

}

**public** **static** **void** paranthesis(String string, **int** open, **int** close) {

// **TODO** Auto-generated method stub

**if** (open == 0 && close == 0) {

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

}

**if** (open > 0) {

*paranthesis*( string + "{",open-1, close+1);

}

**if** (close > 0) {

*paranthesis*(string + "}",open, close-1);

}

}

}

//TestCases

/\*input: 3

\*

\* output: {{{}}}

{{}{}}

{{}}{}

{}{{}}

{}{}{}

input: 2

output: {{}}

{}{}

\*/

/\*

"Given a collection of numbers, return all possible permutations.  
For example,  
[1,2,3] have the following permutations:  
[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1]."

\*/

//PseudoCode

/\*

\* Step1:Iterate for i = start .......last where start = begin index of list, last = end index of list.

\* Step2:Swap integers in first with element in ith index of list.

\* Step3:Call method permute on incrementing the start index of list.

\* Step4:Repeat step 2;

\* Step5:If start == end printout the list.

\*

\*/

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** PermutationOfNumbers {

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

// **TODO** Auto-generated method stub

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

al.add(1);

al.add(2);

al.add(3);

*permute*(al,0,al.size()-1);

}

**public** **static** **void** permute(List<Integer> al,**int** l,**int** r) {

// **TODO** Auto-generated method stub

**if**(l==r){

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

}

**else**{

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

al = *swap*(al,l,i);

*permute*(al,l+1,r);

al = *swap*(al,l,i);

}

}

}

**public** **static** List<Integer> swap(List<Integer> al, **int** l, **int** i) {

// **TODO** Auto-generated method stub

Integer temp = al.get(l);

al.set(l, al.get(i));

al.set(i, temp);

**return** al;

}

}

//Trace

/\*

\* i l r al[0] al[1] al[2]

\* 0 0 2 1 2 3

\* 1 1 2

\* 2 2 2 1 2 3

\* 0 0 2

\* 1 1 2

\* 2 2 2 1 3 2

\* 0 0 2

\* 1 1 2

\* 2 2 2 2 1 3

\* 0 0 2

\* 1 1 2

\* 2 2 2 2 3 1

\* 0 0 2

\* 1 1 2

\* 2 2 2 3 2 1

\* 0 0 2

\* 1 1 2

\* 2 2 2 3 1 2

\*/

//TestCases

/\*

\* input: 1 2 3

\* output:[1, 2, 3]

\* [1, 3, 2]

\* [2, 1, 3]

\* [2, 3, 1]

\* [3, 2, 1]

\* [3, 1, 2]

\*

\* input: 4 5 6

\* output: [4, 5, 6]

\* [4, 6, 5]

\* [5, 4, 6]

\* [5, 6, 4]

\* [6, 5, 4]

\* [6, 4, 5]

\*

\* input: 5 7 8

\* output: [5, 7, 8]

\* [5, 8, 7]

[7, 5, 8]

[7, 8, 5]

[8, 7, 5]

[8, 5, 7]

\*

\*/

/\*

"Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.  
For example, given n = 4,  
[  
[1, 2, 3, 4],   
[12, 13, 14, 5],   
[11, 16, 15, 6],   
[10, 9, 8, 7]  
]"

\*/

//PseudoCode

/\*

\* Step1: Keeping row = rowStart iterate through the column and store the elements in the result and rowStart++.

\* Step2: Keeping col = colEnd iterate through the row and store the elements in the result and colEnd--.

\* Step3: Keeping row = rowEnd iterate through the column and store the elements in the result and rowEnd--.

\* Step4: Keeping col = colStart iterate through the row and store the elements in the result and colStart--.

\* Step5: Repeat steps 1-4 till rowEnd>=0 and colEnd>=0.

\*/

**import** java.util.ArrayList;

**public** **class** SpiralMatrixGenerator {

**static** **int** *rows* =4;

**static** **int** *col* =4;

**static** **int** *rowStart* = 0;

**static** **int** *colStart* =0;

**static** **int** *rowEnd* =3;

**static** **int** *colEnd* =3;

**static** **int** *i*=0;

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

// **TODO** Auto-generated method stub

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

**int** value=4;

**int** max = value\*value;

**for**(**int** m=1;m<=max;m++){

al.add(m);

}

**int**[][] result = **new** **int**[*rows*][*col*];

result = *spiralGenerator*(al,result);

**for**(**int** m=0;m<*rows*;m++){

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

System.***out***.print(result[m][n]);

}

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

}

}

**public** **static** **int**[][] spiralGenerator(ArrayList<Integer> al,**int**[][] result) {

// **TODO** Auto-generated method stub

**while**(*rowEnd*>=0 && *colEnd*>=0){

*fillRow*(al,result);

*rowStart*++;

*fillColumn*(al,result);

*colEnd*--;

*fillRevRow*(al,result);

*rowEnd*--;

*fillRevCol*(al,result);

*colStart*++;

}

**return** result;

}

**public** **static** **int**[][] fillRow(ArrayList<Integer> al, **int**[][] result) {

// **TODO** Auto-generated method stub

**for**(**int** k=*colStart*;k<=*colEnd*;k++){

result[*rowStart*][k] = al.get(*i*);

*i*++;

}

**return** result;

}

**public** **static** **int**[][] fillColumn(ArrayList<Integer> al,**int**[][] result) {

// **TODO** Auto-generated method stub

**for**(**int** k=*rowStart*;k<=*rowEnd*;k++){

result[k][*colEnd*] = al.get(*i*);

*i*++;

}

**return** result;

}

**public** **static** **int**[][] fillRevRow(ArrayList<Integer> al,**int**[][] result) {

// **TODO** Auto-generated method stub

**for**(**int** k=*colEnd*;k>=*colStart*;k--){

result[*rowEnd*][k] = al.get(*i*);

*i*++;

}

**return** result;

}

**public** **static** **int**[][] fillRevCol(ArrayList<Integer> al, **int**[][] result) {

// **TODO** Auto-generated method stub

**for**(**int** k= *rowEnd*;k>=*rowStart*;k--){

result[k][*colStart*] = al.get(*i*);

*i*++;

}

**return** result;

}

}

//Trace

/\*

\* k rowStart rowEnd colStart colEnd result

\* 0 0 3 0 3 1

\* 1 2

\* 2 3

\* 3 4

\* 0 1 3 0 3 5

\* 1 6

\* 2 7

\* 3 8

\* 2 1 3 0 2 9

\* 1 10

\* 0 11

\* 3 1 2 0 2 12

\* 2 13

\* 1 14

\* 0 1 2 1 1 15

\* 1 1 1 1 1 16

\*/

//TestCases

/\*

\*

\* input = 4

\* output = 1234

\* 1213145

\* 1116156

\* 10987

\*

\* input = 3

\* output =123

\* 894

\* 765

\*

\* input = 5

\* output = 123456

\* 20212223247

\* 19323334258

\* 18313635269

\* 173029282710

\* 161514131211

\*

\*

\*/

/\*

Given a sorted array of integers, find the starting and ending position of   
a given target value.   
For example, given [5, 7, 7, 8, 8, 10] and target value 8, return [3, 4].

\*/

//PseudoCode

/\*

\* Step1:Calculate the first occurrence of a character in a string.

\* Step2:Calculate the last occurrence of a character in a string.

\*

\*/

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** StartingAndLastPositionOfValue {

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

// **TODO** Auto-generated method stub

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

al.add(5);

al.add(7);

al.add(7);

al.add(8);

al.add(8);

al.add(10);

**int** value = 8;

List<Integer> result = **new** ArrayList<Integer>();

result.add(al.indexOf(value));

result.add(al.lastIndexOf(value));

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

}

}

//TestCasses

/\*

\* value al al.indexOf(value) al.lastIndexOf(value)

\* 8 [5, 7, 7, 8, 8, 10] 3 4

\* 7 1 2

\*

\*/

/\*

Given a string s, partition s such that every substring of the partition is a palindrome.  
Return all possible palindrome partitioning of s.  
For example, given s = "aab",  
Return  
[  
["aa","b"],  
["a","a","b"]  
]

\*/

//PseudoCode

/\*

\* Step1: First calculate all the substrings of a string.

\* Step2: Split the string using split() method and store it in a string array.

\* Step3: For every string element in the string array reverse the string using stringbuffer

\* and and check if it is a palindrome or not.

\* Step4: If it is palindrome then print the string.

\*/

**public** **class** SubstringPalindrome {

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

String s = "aab";

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

sb = *subString*(s,sb);

*isPalindrome*(sb.toString());

}

**public** **static** **void** isPalindrome(String string) {

// **TODO** Auto-generated method stub

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

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

StringBuffer temp = **new** StringBuffer(s[i]);

temp.reverse();

**if**(temp.toString().equals(s[i]) ){

System.***out***.println("palindrome:" + s[i]);

}

}

}

**public** **static** StringBuffer subString(String s,StringBuffer sb) {

// **TODO** Auto-generated method stub

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

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

sb = sb.append(s.substring(i, j+i));

sb.append(" ");

}

}

**return** sb;

}

}

//Trace

/\*

\* i sb output

\* a aa aab a ab b

\* 0 isPalindrome(a) a

\* 1 isPalindrome(aa) aa

\* 2 isPalindrome(aab)

\* 3 isPalindrome(a) a

\* 4 isPalindrome(ab)

\* 5 isPalindrome(a) b

\*/

//TestCases

/\*

\* input: aab

\* output: a aa a b

\*

\* input: aba

\* output: a aba b a

\*/

/\*

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.  
For example, given the array [−2,1,−3,4,−1,2,1,−5,4], the contiguous subarray[4,−1,2,1] has the largest sum = 6.

\*/

//PseudoCode

/\*

\* Step1: Calculate all the possible subArrays of a given elements.

\* Step2: Calculate sum of all the numbers of all possible subArrays.

\* Step3: Initialize a variable max = a[0] and index=0 ,a = array of sum of each possible subArray.

\* Step4: Iterate through a[] and if any element in a[] is greater than sum for index i, max = a[i] , index =i.

\* Step5: Print element of array of 'index' that is the required array.

\*/

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** SumOfCnSubArr {

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

// **TODO** Auto-generated method stub

List<Object> al = **new** ArrayList<Object>();

List<List<Object>> result = **new** ArrayList<List<Object>>();

al.add(-2);

al.add(1);

al.add(-3);

al.add(4);

al.add(-1);

al.add(2);

al.add(1);

al.add(-5);

al.add(4);

result = *generateSubArray*(al,result);

*sum*(result);

}

**private** **static** **void** sum(List<List<Object>> result) {

// **TODO** Auto-generated method stub

**int** sumT[] = **new** **int**[result.size()];

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

**int** sum =0;

**for**(**int** j=0;j<result.get(i).size();j++){

sum = sum + (**int**)result.get(i).get(j);

sumT[i] = sum;

}

}

*maxSumSubArray*(sumT,result);

}

**public** **static** **void** maxSumSubArray(**int**[] sumT, List<List<Object>> result) {

// **TODO** Auto-generated method stub

**int** max = sumT[0];

**int** index =0;

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

**if**(sumT[i]>max){

max = sumT[i];

index = i;

}

}

System.***out***.println(result.get(index)+ " " + sumT[index]);

}

**public** **static** List<List<Object>> generateSubArray(List<Object> al,List<List<Object>> result) {

// **TODO** Auto-generated method stub

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

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

result.add(al.subList(i, j+1));

}

}

**return** result;

}

}

//Trace

/\*

\* i index max sumT[]

\* 0 1 -1

\* 0 0 1 -4

\* 1 0

\* 2 -1

\* 3 1

\* 4 4 2 2

\* 5 -3

\* 6 1

\* 7 -2

\* 8 2

\* 9 1

\* 10 10 3 3

\* 11 11 4 4

\* 12 -1

\* 13 3

\* 14 1

\* 15 0

\* 16 2

\* 17 3

\* 18 -2

\* 19 2

\* 20 3

\* 21 21 5 5

\* 22 22 6 6

\* 23 1

\* 24 5

\* 25 1

\* 26 2

\* 27 -3

\* 28 1

\* 29 3

\* 30 -2

\* 31 2

\* 32 -4

\* 33 0

\* 34 -1

\*/

//TestCases

/\*

\* input: [−2,1,−3,4,−1,2,1,−5,4]

\* output: [4, -1, 2, 1] 6

\*

\* input:[−1,2,1,−5,4]

\* output: [2, 1] 3

\*/