public class R\_3\_1 {  
 private static int *a*=12;  
 private static int *b*=5;  
 private static int *n*=100;  
 private static int *cur*=92;  
  
 public static void main(String[] args) {  
 for (int i = 0; i < 5; i++) {  
 *cur*=(*a*\* *cur* +*b*)%*n*;  
 System.*out*.println(*cur*);  
  
 }  
 }  
}  
//3 R-3.3 Explain the changes that would have to be made to the program of Code Fragment 3.8 so that it could perform the Caesar cipher for messages that are written in an alphabet-based language other than English, such as Greek, Russian, or Hebrew.  
//  
//  
//  
//الشرح  
//1-الابجديه تقوم بتعديل ALPHABETليمثل لغه معينع  
//2-اعين الحرةف اسوي تحدجيث بطريقه shiftByللتعامل مع ابجديه  
//3- الحساب الحسابي المعدد تعديل طريقه shiftAmountلتنفيذ الحساب الحسابي الابجدي  
//4- معالجه الادخال والإخراج بضبط معالجه الادخال و الإخراج وذللك لدعم اللغه وتاكد اذا كان يعمل بشكل صحيح  
//وذلك م بنسويه بشكل عملي ب الكود ب البرنامج من عمل شفره وفكهه  
//  
//  
//R-3.13 What is the difference between a shallow equality test and a deep equality test between two Java arrays, A and B, if they are one-dimensional arrays of type int? What if the arrays are two-dimensional arrays of type int?  
//حل السوال  
//1. اختبار المساواة الضحلة (Shallow Equality):  
//عند اختبار المساواة الضحلة بين مصفوفتين، يتم فحص ما إذا كانت المرجعين للمصفوفتين هما نفس المرجع أو يشيران إلى نفس المكان في الذاكرة. بمعنى آخر، إذا كانت المتغيرات A و B تشيران إلى نفس المصفوفة، فإن اختبار المساواة الضحلة سيكون صحيحًا. ولا يتم فحص محتوى المصفوفات بشكل فعلي.  
//  
//- في حالة المصفوفات ذات البعد الواحد (one-dimensional arrays):  
//اختبار المساواة الضحلة بين المصفوفتين بيكون صح  
//int[] A = {1, 2, 3};  
//int[] B = A;  
//  
//boolean shallowEquality = (A == B); بتكون true  
//  
//  
//- في حالة المصفوفات ذات البعدين (two-dimensional arrays):  
//اختبار المساواة الضحلة ليس مفيدًا في حالة المصفوفات ذات البعدين، حيث أن المتغيرات A و B في هذه الحالة تشيران إلى مصفوفتين منفصلتين في الذاكرة، حتى إذا كانت المحتويات متطابقة. لذا، اختبار المساواة الضحلة سيكون مش صح  
//int[][] A = {{1, 2}, {3, 4}};  
//int[][] B = A;  
//  
//boolean shallowEquality = (A ==بتكونfalse  
//  
//  
//2. اختبار المساواة العميقة (Deep Equality):  
//عند اختبار المساواة العميقة بين مصفوفتين، يتم فحص الشي الفعلي للمصفوفتين بشكل متكامل. ولازم أن يحتوي كل عنصر في المصفوفة A على نفس القيمة والترتيب مع مصفوفة B.  
//  
//- في حالة المصفوفات ذات البعد الواحد (one-dimensional arrays):  
//اختبار المساواة العميقة يتطلب مقارنة كل عنصر في المصفوفتين كل وحده وحدها يعني منفصل  
//int[] A = {1, 2, 3};  
//int[] B = {1, 2, 3};  
//  
//boolean deepEquality= Arrays.equals(A, B); بتكون true  
//```  
//  
//- في حالة المصفوفات ذات البعدين (two-dimensional arrays):  
//اختبار المساواة العميقة يتطلب مقارنة كل عنصر في المصفوفتين كل واحد وحده منفصل  
//int[][] A = {{1, 2}, {3, 4}};  
//int[][] B = {{1, 2}, {3, 4}};  
//  
//boolean deepEquality = Arrays.deepEquals(A, B) بتكون صح true  
//  
//  
//  
//  
//R-3-14Give three different examples of a single Java statement that assigns variable, backup, to a new array with copies of all int entries of an existing array, original.  
//حل السوال  
//هنا ثلاثة أمثلة مختلفة الاوله اللي هي  
//1. استخدام حلقة for لنسخ العناصر يدويًا: زي الكود ذا  
//int[] backup = new int[original.length];  
//for (int i = 0; i < original.length; i++) {  
// backup[i] = original[i];  
//}  
//  
//  
//تبدأ العبارة بإنشاء مصفوفة جديدة بنفس الحجم مثل `original`، بعدين تستخدم حلقة `for` للانتقال عبر المصفوفة الأصلية ونسخ قيم كل عنصر في `backup` بشكل منفصل.  
//  
//2. استخدام دالة `System.arraycopy()`:  
//  
//  
//int[] backup = new int[original.length];  
//System.arraycopy(original, 0, backup, 0, original.length);  
//  
//  
//تبدأ العبارة بإنشاء مصفوفة جديدة backup بنفس الحجم مثل original، بعدين تستخدم دالة System.arraycopy() لنسخ قيم المصفوفة الأصلية إلى backup. الدالة arraycopy() تستخدم لنسخ مجموعة من العناصر بين مصفوفتين.  
//  
//3. استخدام دالة Arrays.copyOf():  
//  
//  
//int[] backup = Arrays.copyOf(original, original.length);  
//  
//  
//  
//C-3.17 Let A be an array of size n ≥ 2 containing integers from 1 to n−1 inclusive, one of which is repeated. Describe an algorithm for finding the integer in A that is repeated.  
//حل السوال  
//1. قم بإنشاء مصفوفة ثنائية البعد تحتوي على n عنصرًا، حيث تكون الأعمدة تمثل القيم من 1 إلى n-1 والصفوف تمثل المواضع في المصفوفة A.  
//2. قم بتكرار العناصر في المصفوفة A وضعها في المواضع المناسبة في المصفوفة الثنائية البعد.  
//3. خلال العملية، عندما تجد عنصرًا يحتل موقعًا بالفعل في المصفوفة الثنائية البعدي، فهذا يعني أنه تم تكراره في المصفوفة الأصلية.  
//4. قم بإرجاع العنصر المكرر.  
//  
//  
//والكود في البرنامج  
//  
//C-3.18 Let B be an array of size n ≥ 6 containing integers from 1 to n−5 inclusive, five of which are repeated. Describe an algorithm for finding the five integers in B that are repeated.  
//لحل السوال  
//1. قم بإنشاء خريطة (Map) لتتبع تكرار الأعداد. المفتاح في الخريطة سيكون العدد الموجود في المصفوفة B، والقيمة ستكون عدد التكرارات الموجودة لكل عدد.  
//2. قم بتكرار عناصر المصفوفة B وزيادة عدد التكرار لكل عدد في الخريطة.  
//3. بعد الانتهاء من العبور عبر المصفوفة B، قم بفحص الخريطة للعثور على الأعداد التي لديها عدد تكرارات يساوي 5.  
//4. قم بتخزين الأعداد المكررة الخمس في مصفوفة أخرى أو هيكل بيانات مناسب.  
//  
//والكود في البرنامج  
//c-3-20 Give examples of values for a and b in the pseudorandom generator given on page 113 of this chapter such that the result is not very random looking, for n = 1000.  
//يتم توفير مُولِّد الأعداد العشوائية التكراري المذكور في صفحة 113 من الفصل بواسطة الصيغة التالية:  
//  
//xi = (a \* xi-1 + b) mod n  
//  
//لتوليد قيم غير عشوائية بشكل واضح لـ n = 1000، نحتاج إلى اختيار قيم لـ a و b تؤدي إلى نمط متكرر أو تسلسل قابل للتنبؤ. فيما يلي بعض الأمثلة:  
//  
//مثال 1:  
//a = 0  
//b = 0  
//  
//في هذا المثال، تتقلص الصيغة إلى:  
//  
//xi = (0 \* xi-1 + 0) mod 1000  
//  
//تعطينا هذه الصيغة نفس القيمة 0 بغض النظر عن القيمة السابقة xi-1. بالتالي، ستكون جميع القيم المولدة هي 0.  
//  
//مثال 2:  
//a = 1  
//b = 1  
//  
//في هذا المثال، تصبح الصيغة:  
//  
//xi = (1 \* xi-1 + 1) mod 1000  
//  
//ستكون القيم المولدة في هذه الحالة تتكرر بنمط محدد. على سبيل المثال، إذا كانت القيمة الأولية xi-1 تساوي 0، فسيكون لدينا التالي:  
//  
//x0 = (1 \* 0 + 1) mod 1000 = 1  
//x1 = (1 \* 1 + 1) mod 1000 = 2  
//x2 = (1 \* 2 + 1) mod 1000 = 3  
//x3 = (1 \* 3 + 1) mod 1000 = 4  
//...  
//وهكذا تستمر القيم في التكرار بنمط متكرر.  
//  
//مثال 3:  
//a = 2  
//b = 3  
//  
//في هذا الثل، يتحول المولد إلى:  
//  
//xi = (2 \* xi-1 + 3) mod 1000  
//  
//بيتكون القيم المولدة في هذه الحالة كمان تكرارية بنمط محدد. مثال، إذا كانت القيمة الأولية xi-1 تساوي 0، فسنحصل على التالي:  
//  
//x0 = (2 \* 0 + 3) mod 1000 = 3  
//x1 = (2 \* 3 + 3) mod 1000 = 9  
//x2 = (2 \* 9 + 3) mod 1000 = 21  
//x3 = (2 \* 21 + 3) mod 1000 = 45  
//...  
//وبالتالي، ستكون القيم المولدة تتكرر قيم مححده  
//C-3.21 Suppose you are given an array, A, containing 100 integers that were generated using the method r.nextInt(10), where r is an object of type java.util.Random. Let x denote the product of the integers in A. There is a single number that x will equal with probability at least 0.99. What is that number and what is a formula describing the probability that x is equal to that number?  
//.  
//  
//لحساب حاصل الضرب x، ضرب جميع الأعداد في معًا. يمكننا استخدام الصيغة التالية لحساب x:  
//  
//x = A[0] \* A[1] \* A[2] \* ... \* A[99]  
//  
//حيث A[i] يمثل العدد الصحيح في الموضع i من المصفوفة A.  
//  
//الآن، هناك عدد واحد فقط يساوي x بنسبة احتمال لا تقل عن 0.99. هذا العدد هو الصفر (0). سبب ذلك هو أنه إذا كان أحد الأعداد في المصفوفة A يساوي الصفر، فإن حاصل الضرب x سيكون صفرًا بالتأكيد. ونظرًا لأن الأعداد تم توليدها بتوزيع عشوائي متساوٍ وتتراوح بين 0 و 9، فإن الاحتمالية التي يكون فيها أحد الأعداد يساوي الصفر هي 1/10 أو 0.1.  
//  
//بالتالي، الصيغة التي تصف الاحتمالية التي يكون فيها x يساوي الصفر هي:  
//  
//P(x = 0) = 0.1  
//  
//وبما أن هناك عدد واحد فقط يساوي x بنسبة احتمال لا تقل عن 0.99، فإذا افترضنا أن هذا العدد ليس الصفر، فإن الاحتمالية الأدنى له هي:  
//  
//P(x = non-zero number) ≥ 0.99 - P(x = 0) ≥ 0.99 - 0.1 = 0.89  
//  
//إذاً، هناك عدد واحد على الأقل يساوي x بنسبة احتمال لا تقل عن 0.89.  
//C-3.23 Suppose you are designing a multiplayer game that has n ≥ 1000 players, numbered 1 to n, interacting in an enchanted forest. The winner of this game is the first player who can meet all the other players at least once (ties are allowed). Assuming that there is a method meet(i, j), which is called each time a player i meets a player j (with i 6= j), describe a way to keep track of the pairs of meeting players and who is the winner.  
//  
//  
//لتتبع أزواج اللاعبين الذين يلتقون وتحديد الفائز في لعبة متعددة اللاعبين، يمكن استخدام هيكل بيانات يسمى "مصفوفة الاجتماعات" (Meeting Matrix) ومتغير لتتبع الفائز.  
//  
//المصفوفة الاجتماعية هي مصفوفة مربعة ثنائية الأبعاد بحجم n × n، حيث n هو عدد اللاعبين في اللعبة. ستتمثل قيم عناصر المصفوفة في الترتيبات التي يلتقي فيها اللاعبون. سنستخدم 1 لتمثيل اللقاء بين اللاعبين، و 0 لتمثيل عدم اللقاء.  
//  
//بدايةً، يتم تهيئة المصفوفة الاجتماعية بقيم افتراضية 0 لجميع العناصر، مما يعني أنه لم يتم لقاء أي لاعب حتى الآن.  
//  
//عندما يلتقي لاعب i بلاعب j، سنقوم بتحديث المصفوفة الاجتماعية عن طريق تعيين القيمة 1 في العنصر (i, j) و (j, i) من المصفوفة، وذلك للإشارة إلى أن اللاعبين i و j قد التقيا.  
//  
//بعد كل اجتماع جديد، يجب التحقق مما إذا كان اللاعب الفائز قد تم تحقيقه. يتم تتبع حالة اللاعبين وعدد لقاءاتهم باستخدام متغير يسمى "عداد اللقاءات" (Meetings Counter). في البداية، سيكون قيم عداد اللقاءات لجميع اللاعبين هو 0.  
//  
//عند كل اجتماع جديد بين اللاعبين i و j، سنقوم بزيادة قيم عداد اللقاءات بواحد لكل من اللاعبين i و j. ثم سنحقق ما إذا كان أي لاعب قد تحقق من شروط الفوز، وهي أن يكون عدد لقاءاته يساوي n - 1.  
//  
//إذا كان أحد اللاعبين يلبي شروط الفوز، فسيتم اعتباره الفائز. يمكننا تخزين رقم اللاعب الفائز في متغير فردي يسمى "اللاعب الفائز" (Winner Player).  
//  
//  
//java.util Methods for Arrays  
//fill(A, x)  
//copyOf(A, n)  
//copyOfRange(A, s, t):  
//toString(A)  
//sort(A):  
//binarySearch(A, x)  
//حل السوال  
//1. fill(A, x):  
//هذه الطريقة تقوم بتعبئة جميع عناصر المصفوفة A بقيمة محددة x. يتم تعديل المصفوفة A مباشرة دون إنشاء مصفوفة جديدة.  
//  
//2. copyOf(A, n):  
//تستخدم هذه الطريقة لنسخ المصفوفة A إلى مصفوفة جديدة بحجم n. إذا كان n أكبر من طول المصفوفة الأصلية A، ستتم ملء الفراغات بقيم ا  
// فتراضية (0 للأعداد الصحيحة و null للكائنات المرجعية). إذا كان n أصغر من طول A، ستتم قص المصفوفة A لتناسب الحجم الجديد.  
//  
//3. copyOfRange(A, s, t):  
//تستخدم هذه الطريقة لنسخ جزء محدد  
// من المصفوفة A بين الفهرسين s و t (باستثناء t). ستنشئ مصفوفة جديدة تحتوي على النطاق المحدد من المصفوفة الأصلية A.  
//  
//4. toString(A):  
//تستخدم هذه  
// الطريقة لإرجاع سلسلة نصية تمثل المصفوفة A. يتم تمثيل العناصر الموجودة في المصفوفة بواسطة سلاسل نصية، مفصولة بفواصل.  
//  
//5. sort(A):  
//تستخدم هذه الطريقة لفرز المصفوفة A بترتيب تصاعدي. إذا كانت العناصر في المصفوفة من نوع Comparable، ستتم مقارنتها  
// وفرزها وفقًا للترتيب الطبيعي. إذا كانت العناصر غير قابلة للمقارنة، فسيتم رمي استثناء من نوع ClassCastException.  
//  
//6. binarySearch(A, x):  
//تستخدم هذه الطريقة للبحث عن قيمة x في المصفوفة A المرتبة. إذا تم العثور على القيمة، فستُرجع موقعها في المصفوفة

//

import java.util.Arrays;  
public class R\_3\_2 {  
 public static int[] removeRandomEntries(int[]array) {  
 int[] newArray = new int[array.length];  
 System.*arraycopy*(array, 0, newArray, 0, array.length);  
 while (newArray.length > 0) {  
 int randomIndex = (int) (Math.*random*() \* newArray.length);  
 System.*arraycopy*(newArray, randomIndex + 1, newArray, randomIndex, newArray.length - randomIndex - 1);  
 array = Arrays.*copyOf*(array, array.length - 1);  
 newArray=Arrays.*copyOf*(newArray,newArray.length-1);  
  
 }  
 return newArray;  
 }  
  
 public static void main(String[] args) {  
 int[] array={1,2,3,4,5};  
 int [] result= *removeRandomEntries*(array);  
 System.*out*.println(Arrays.*toString*(result));  
 }  
}

//

import java.util.Random;  
public class R\_3\_2l {  
 public static <A> void removeRandomElements(A[]array){  
 Random random=new Random();  
 int remaining=array.length;  
 while (remaining>0){  
 int index=random.nextInt(remaining);  
 A temp =array[index];  
 array[remaining-1]=temp;  
 System.*out*.println("Removed:"+temp);  
 remaining--;  
 }  
 }  
  
 public static void main(String[] args) {  
 Integer[]array={1,2,3,4,5,6};  
 *removeRandomElements*(array);  
 }  
}

//

import java.util.HashMap;  
import java.util.Map;  
public class R\_3\_3 {  
 public static final String *ALPHABET* ="ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
 public static final int *SHIFT\_AMOUNT* = 3;  
 private static Map<Character,Integer>*charToIndex*;  
 private static char[]*indexTOChar*;  
 static {  
 *charToIndex*=new HashMap<>();  
 *indexTOChar*=new char[*ALPHABET*.length()];  
 for (int i = 0; i <*ALPHABET*.length(); i++) {  
 *charToIndex*.put(*ALPHABET*.charAt(i),i);  
 *indexTOChar*[i]=*ALPHABET*.charAt(i);  
  
 }  
 }  
 public static String encrypt(String message){  
 StringBuilder encryptedMassage=new StringBuilder();  
 for (char c:message.toCharArray()){  
 if (*charToIndex*.containsKey(Character.*toUpperCase*(c))){  
 int index=*charToIndex*.get(Character.*toUpperCase*(c));  
 int encryptedIndex= (index+*SHIFT\_AMOUNT*)%*ALPHABET*.length();  
 char encryptedChar=*indexTOChar*[encryptedIndex];  
 encryptedMassage.append(Character.*isLowerCase*(c)? Character.*toLowerCase*(encryptedChar):encryptedChar);  
 }  
 else {  
 encryptedMassage.append(c);  
 }  
 }  
 return encryptedMassage.toString();  
  
 }  
 public static String decrypt(String massage){  
 StringBuilder decryptedMessage=new StringBuilder();  
 for (char c:massage.toCharArray()){  
 if (*charToIndex*.containsKey(Character.*toUpperCase*(c))){  
 int index = *charToIndex*.get(Character.*toUpperCase*(c));  
 int decryptedIndex=(index-*SHIFT\_AMOUNT*+*ALPHABET*.length())%*ALPHABET*.length();  
 char decryptedChar=*indexTOChar*[decryptedIndex];  
 decryptedMessage.append(Character.*isLowerCase*(c)? Character.*toLowerCase*(decryptedChar):decryptedChar);  
 }  
 else {  
 decryptedMessage.append(c);  
 }  
  
 }  
 return decryptedMessage.toString();  
 }  
  
 public static void main(String[] args) {  
 String message="HELLO WORLD";  
 String encryptedMessage=*encrypt*(message);  
 String decryptedMessage=*decrypt*(encryptedMessage);  
 System.*out*.println("Original message: "+message);  
 System.*out*.println("Encrypted massage: " +encryptedMessage);  
 System.*out*.println("Decrypted massage: " +decryptedMessage);  
 }  
}

//

public class R\_3\_4 {  
 private static final int *SIZE* = 3;  
 private char[][] board;  
 private char currentPlayerMark;  
  
 public R\_3\_4() {  
 board = new char[*SIZE*][*SIZE*];  
 currentPlayerMark = 'X';  
 initializeBoard();  
 }  
  
 private void initializeBoard() {  
 for (int i = 0; i < *SIZE*; i++) {  
 for (int j = 0; j < *SIZE*; j++) {  
 board[i][j] = '-';  
 }  
 }  
 }  
  
 public void printBoard() {  
 System.*out*.println("-------------");  
 for (int i = 0; i < *SIZE*; i++) {  
 System.*out*.print("| ");  
 for (int j = 0; j < *SIZE*; j++) {  
 System.*out*.print(board[i][j] + " | ");  
 }  
 System.*out*.println();  
 System.*out*.println("-------------");  
 }  
 }  
  
 public boolean isBoardFull() {  
 for (int i = 0; i < *SIZE*; i++) {  
 for (int j = 0; j < *SIZE*; j++) {  
 if (board[i][j] == '-') {  
 return false;  
 }  
 }  
 }  
 return true;  
 }  
  
 public boolean checkForWin() {  
 return checkRowsForWin() || checkColumnsForWin() || checkDiagonalsForWin();  
 }  
  
 private boolean checkRowsForWin() {  
 for (int i = 0; i < *SIZE*; i++) {  
 if (checkRowCol(board[i][0], board[i][1], board[i][2])) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private boolean checkColumnsForWin() {  
 for (int i = 0; i < *SIZE*; i++) {  
 if (checkRowCol(board[0][i], board[1][i], board[2][i])) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private boolean checkDiagonalsForWin() {  
 return (checkRowCol(board[0][0], board[1][1], board[2][2]) ||  
 checkRowCol(board[0][2], board[1][1], board[2][0]));  
 }  
  
 private boolean checkRowCol(char c1, char c2, char c3) {  
 return (c1 != '-' && c1 == c2 && c2 == c3);  
 }  
  
 public void putMark(int row, int col) {  
 if (row < 0 || row >= *SIZE* || col < 0 || col >= *SIZE*) {  
 throw new IllegalArgumentException("Invalid row or column!");  
 }  
  
 if (board[row][col] != '-') {  
 throw new IllegalStateException("Cell is already occupied!");  
 }  
  
 board[row][col] = currentPlayerMark;  
 }  
  
 public void changePlayer() {  
 currentPlayerMark = (currentPlayerMark == 'X') ? 'O' : 'X';  
 }  
  
 public char getCurrentPlayerMark() {  
 return currentPlayerMark;  
 }  
  
 public static void main(String[] args) {  
 R\_3\_4 game = new R\_3\_4();  
 game.putMark(0, 0);  
 game.putMark(0, 1);  
 game.putMark(1, 1);  
 game.putMark(1, 0);  
 game.putMark(2, 2);  
  
 // Trying to put mark after the game has already been won  
 try {  
 game.putMark(2, 0);  
 } catch (IllegalStateException e) {  
 System.*out*.println(e.getMessage());  
 }  
  
 game.printBoard();  
 }  
}

//

public class R\_3\_17 {  
 int findDuplicate(int[] A) {  
 int n = A.length;  
 int[] count = new int[n]; // المصفوفة الثنائية البعد  
  
 for (int i = 0; i < n; i++) {  
 int num = A[i];  
 if (count[num] > 0) {  
 return num; // تم العثور على العنصر المكرر  
 } else {  
 count[num]++; // زيادة العداد في الموقع المناسب  
 }  
 }  
  
 return -1;  
 }  
}  
//شرح الكود في الورد وحل السوال R-3.13,R-3-14في الورد

//

public class Scoreboard {  
 private int numEntries = 0;  
 private GameEntry[] board;  
  
 public Scoreboard(int capacity) {  
 board = new GameEntry[capacity];  
 }  
  
 public void add(GameEntry entry) {  
 int newScore = entry.getScore();  
  
 if (numEntries < board.length || newScore > board[numEntries - 1].getScore()) {  
 if (numEntries < board.length) {  
 numEntries++;  
 }  
  
 int i = numEntries - 1;  
  
 while (i > 0 && newScore > board[i - 1].getScore()) {  
 board[i] = board[i - 1];  
 i--;  
 }  
  
 board[i] = entry;  
 }  
 }  
  
 public GameEntry remove(int index) {  
 if (index < 0 || index >= numEntries) {  
 throw new IndexOutOfBoundsException("Invalid index: " + index);  
 }  
  
 GameEntry removedEntry = board[index];  
  
 for (int i = index; i < numEntries - 1; i++) {  
 board[i] = board[i + 1];  
 }  
  
 board[numEntries - 1] = null;  
 numEntries--;  
  
 return removedEntry;  
 }  
}  
```

//

import java.util.HashMap;  
import java.util.Map;  
public class C\_3\_18{  
  
  
int[] findRepeatedIntegers(int[] B) {  
 int n = B.length;  
 Map<Integer, Integer> frequencyMap = new HashMap<>(); // خريطة التكرارات  
  
 for (int i = 0; i < n; i++) {  
 int num = B[i];  
 frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1); // زيادة تكرار العدد في الخريطة  
 }  
  
 int[] repeatedIntegers = new int[5];  
 int index = 0;  
  
 for (Map.Entry<Integer, Integer> entry : frequencyMap.entrySet()) {  
 if (entry.getValue() == 5) {  
 repeatedIntegers[index] = entry.getKey(); // تخزين العدد المكرر في المصفوفة  
 index++;  
 }  
 }  
  
 return repeatedIntegers;  
}  
}  
//شرح تكليف ب الرود

//

public class C\_3\_19 {  
  
 private int numEntries;  
 private GameEntry[] board;  
  
 public C\_3\_19(int capacity) {  
 board = new GameEntry[capacity];  
 }  
  
 public void add(GameEntry e) {  
 if (numEntries < board.length) {  
 board[numEntries++] = e;  
 }  
 }  
  
 public GameEntry remove(int i) throws IndexOutOfBoundsException {  
 if (i < 0 || i >= numEntries) {  
 throw new IndexOutOfBoundsException("Invalid index: " + i);  
 }  
 GameEntry temp = board[i];  
 board[i] = board[numEntries - 1];  
 board[numEntries - 1] = null;  
 numEntries--;  
 return temp;  
 }  
  
}

//

import java.util.Random;  
  
public class C\_3\_22 {  
 public static void shuffle(int[] A) {  
 Random random = new Random();  
  
 for (int i = A.length - 1; i > 0; i--) {  
 int j = random.nextInt(i + 1); // عدد عشوائي بين 0 و i  
  
 // تبديل العنصر A[i] مع العنصر A[j]  
 int temp = A[i];  
 A[i] = A[j];  
 A[j] = temp;  
 }  
 }  
  
 public static void main(String[] args) {  
 int[] array = {1, 2, 3, 4, 5};  
 *shuffle*(array);  
  
 System.*out*.println("Shuffled Array:");  
 for (int num : array) {  
 System.*out*.print(num + " ");  
 }  
 }  
}

//

public class C\_3\_23 {  
 private int[][] meetingMatrix;  
 private int[] meetingsCounter;  
 private int winnerPlayer;  
  
 public C\_3\_23(int numPlayers) {  
 meetingMatrix = new int[numPlayers][numPlayers];  
 meetingsCounter = new int[numPlayers];  
 winnerPlayer = -1;  
 }  
  
 public void meet(int player1, int player2) {  
 meetingMatrix[player1][player2] = 1;  
 meetingMatrix[player2][player1] = 1;  
  
 meetingsCounter[player1]++;  
 meetingsCounter[player2]++;  
  
 if (meetingsCounter[player1] == meetingMatrix.length - 1) {  
 winnerPlayer = player1;  
 } else if (meetingsCounter[player2] ==meetingMatrix.length - 1) {  
 winnerPlayer = player2;  
 }  
 }  
  
 public int getWinner() {  
 return winnerPlayer;  
 }  
  
 public static void main(String[] args) {  
 int numPlayers = 1000;  
 C\_3\_23 game = new C\_3\_23(numPlayers);  
  
 // عين أمثلة للاعبين الذين يلتقون  
 game.meet(1, 2);  
 game.meet(1, 3);  
 game.meet(2, 3);  
 game.meet(4, 5);  
 // ...  
  
 int winner = game.getWinner();  
 if (winner != -1) {  
 System.*out*.println("Player " + winner + " is the winner!");  
 } else {  
 System.*out*.println("No winner yet.");  
 }  
 }  
}  
//مثال بسيط وشرح في الورد

//

public class C\_3\_24 {  
  
 public static int[][][] addArrays(int[][][] array1, int[][][] array2) {  
 int length = array1.length;  
 int width = array1[0].length;  
 int height = array1[0][0].length;  
  
 int[][][] result = new int[length][width][height];  
  
 for (int i = 0; i < length; i++) {  
 for (int j = 0; j < width; j++) {  
 for (int k = 0; k < height; k++) {  
 result[i][j][k] = array1[i][j][k] + array2[i][j][k];  
 }  
 }  
 }  
  
 return result;  
 }  
 }