Data Structure Lab3 -Arrays

Exercises and Homework

java.util Methods for Arrays

fill(A, x) This method fills the entire specified array (A) with the value (x).

copyOf(A, n*) This method creates a new array with the same element type as the original array (A) and a length of (n). It then copies the elements from the original array to the new array, truncating or padding with default values as necessary.*

copyOfRange(A, s, t): This method creates a new array with the same element type as the original array (A) and a length of (t - s). It then copies the elements from the specified range (s, inclusive) to (t, exclusive) of the original array to the new array.

toString(A*) This method converts the specified array (A) to a string representation. The format of the string representation depends on the type of the elements in the array.*

sort(A*): This method sorts the elements in the specified array (A) in ascending order. The elements are compared using their natural ordering, which means that the elements must be comparable.*

binarySearch(A, x*) This method searches for the specified value (x) in the sorted array (A) using the binary search algorithm. The method returns the index of the element if it is found, or a negative value if it is not found.*// Fill an array with the value 5

```
int[] a = new int[10];

Arrays.fill(a, 5);


// Create a copy of an array

int[] b = Arrays.copyOf(a, 5);


// Create a copy of a portion of an array

int[] c = Arrays.copyOfRange(a, 2, 7);


// Convert an array to a string

String str = Arrays.toString(a);
```

```
// Sort an array
Arrays.sort(a);

// Search for a value in an array
int index = Arrays.binarySearch(a, 10);
```

1	R-3.1	Give the next five pseudorandom numbers generated by the process described on page 113, with a = 12, b = 5, and n = 100, and 92 as the seed for cur. See page 113 باللغة العربية، الأرقام العشوائية الزائفة التالية التي تم إنشاؤها بواسطة العملية الموضحة في الصفحة ١١٣ هي: • 9 • 13 • 61 • 37 • 49 العملية تستند إلى خوارزمية مولد الأرقام العشوائية الخطية، وهي نوع من مولدات الأرقام العشوائية الزائفة التي تستخدم الصيغة التالية: $cur=(a\times prev+b) \bmod n$ حيث cur هو الرقم العشوائي الزائف الحالي، prev هو الرقم العشوائي الزائف السابق، a و b هما ثوابت، و n هو المضاعف. البذرة هي القيمة الأولية لـ prev. لإنشاء الرقم العشوائي الزائف التالي، نستخدم قيم prev و n و b و a في الصيغة ونحسب cur. على سبيل المثال، لإنشاء الرقم prev كـ cur ثم نكرر العملية مع القيمة الجديدة لـ prev. ونحصل على prev العشوائي الزائف الأول، نستخدم البذرة ٩٢ كـ: $cur=(12\times 92+5) \bmod 100=9$ ثم نستخدم ٩ كـ prev ونحصل على جديد: $cur=(12\times 9+5) \bmod 100=13$ وهكذا، حتى نحصل على خمسة أرقام عشوائية زائفة.

<table>
<tr>
<td>2</td>
<td>R-3.2</td>
<td>Write a Java method that repeatedly selects and removes a random entry from an array until the array holds no more entries.

```
public class Q2 {
    import java.util.Random;

    public class ArrayRandomEmpty {

        // عناصرها وتزيل نوع أي من مصفوفة تأخذ طريقة
عشوائيًا
        public static <T> void removeRandomElements(T[]
array) {
            // عشوائية مؤشرات لإنشاء عشوائي كائن إنشاء
            Random random = new Random();
            // المصفوفة في المتبقية العناصر عدد تتبع
            int remaining = array.length;
            // فارغة المصفوفة تكون حتى كرر
            while (remaining > 0) {
                // العناصر نطاق ضمن عشوائي مؤشر توليد
المتبقية
                int index = random.nextInt(remaining);
                // آخر مع العشوائي المؤشر في العنصر مبادلة
متبقي عنصر
                T temp = array[index];
                array[index] = array[remaining - 1];
                array[remaining - 1] = temp;
                // المزال العنصر طباعة
                System.out.println("Removed: " + temp);
                // بواحد المتبقية العناصر عدد خفض
                remaining--;
            }
        }

        // من مصفوفة مثال مع الطريقة لاختبار رئيسية طريقة
الصحيحة الأعداد
        public static void main(String[] args) {
            Integer[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9};
            removeRandomElements(array);
        }
    }
}
```

</td>
</tr>
<tr>
<td>3</td>
<td>R-3.3</td>
<td>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.

كود القطعة ٣,٨ هو برنامج جافا ينفذ تشفير قيصر للرسائل المكتوبة باللغة الإنجليزية. يستخدم رموز</td>
</tr>
</table>

كما يفترض أن قيمة الانتقال بين ٠ و ٢٥ ، وأن .للحروف لإجراء التشفير وفك التشفير ASCII
الرسالة تحتوي فقط على أحرف كبيرة[1]

لتعديل البرنامج ليعمل مع لغات أخرى تستند إلى الأبجدية ، مثل اليونانية أو الروسية أو العبرية ، نحتاج إلى إجراء التغييرات التالية:

- أو UTF-8 استخدم نظام ترميز حروف مختلف يدعم اللغة المطلوبة ، مثل
سيسمح هذا لنا بتمثيل الحروف في اللغة كأرقام يمكن التلاعب بها بواسطة .Unicode
خوارزمية التشفير[2]
- على سبيل المثال ، إذا كانت اللغة .ضبط نطاق قيمة الانتقال لتتناسب مع حجم أبجدية اللغة
تحتوي على ٣٣ حرفًا ، فيجب أن تكون قيمة الانتقال بين ٠ و ٣٢[3]
- التعامل مع الحالة التي تحتوي فيها الرسالة على أحرف صغيرة أو علامات ترقيم أو رموز أخرى ليست جزءًا من الأبجدية. إحدى الطرق الممكنة هي تجاهلها وتركها دون تغيير في طريقة أخرى ممكنة هي تعيينها إلى بعض القيم المحددة مسبقًا يمكن عكسها .الإخراج بواسطة خوارزمية فك التشفير

```
public class Q3 {


            import java.util.Scanner;

    public class CaesarCipher {

        // التشفير وظيفة
        public static String encrypt(String message, int
shift) {
            String cipher = "";
            for (int i = 0; i < message.length(); i++) {
                // في للحرف يونيكود الكود نقطة على احصل
i الموضع
                int code = message.codePointAt(i);
                // يونانيًا حرفًا الحرف كان إذا مما تحقق
كبيرًا
                if (code >= 913 && code <= 937) {
                    // الأبجدية حول ولف الانتقال طبق
                    code = ((code - 913 + shift) % 24) +
913;
                }
                // يونانيًا حرفًا الحرف كان إذا مما تحقق
صغيرًا
                else if (code >= 945 && code <= 969) {
                    // الأبجدية حول ولف الانتقال طبق
                    code = ((code - 945 + shift) % 24) +
945;
                }
                // تغيير دون الحرف اترك ، ذلك خلاف
                // التشفير بنص الحرف ألحق
                cipher += Character.toString(code);
            }
            return cipher;
        }

        // التشفير فك وظيفة
        public static String decrypt(String cipher, int
shift) {
```

| | | ```
String message = "";
for (int i = 0; i < cipher.length(); i++) {
    // احصل على نقطة الكود يونيكود في للحرف الموضع i
    int code = cipher.codePointAt(i);
}
```<br>• |
| 4 | R-3.4 | The TicTacToe class of Code Fragments 3.9 and 3.10 has a flaw, in that it allows a player to place a mark even after the game has already been won by someone. Modify the class so that the putMark method throws an IllegalStateException in that case<br>```
public class Q4 {


    public class TicTacToe {
        // القطعة كود في كما ومتغيرات ثوابت 3.9
        public void putMark(int i, int j) throws IllegalArgumentException, IllegalStateException {
            if ((i < 0) || (i > 2) || (j < 0) || (j > 2))
                throw new IllegalArgumentException("موضع صالح غير اللوح");
            if (board[i][j] != EMPTY)
                throw new IllegalArgumentException("موضع مشغول اللوح");
            if (winner() != EMPTY)
                throw new IllegalStateException("فاز لقد بالفعل اللعبة");
            board[i][j] = player; // وضع العلامة للاعب الحالي
            player = -player; // اللاعبين تبديل (يستخدم أن حقيقة X = - O)
        }
        // القطعة كود في كما الفئة بقية 3.10
    }
```<br>التغيير الرئيسي هو إضافة شرط للتحقق مما إذا كان اللعبة قد فازت بها بالفعل من خلال استدعاء طريقة الفائز () ، والتي تعيد علامة الفائز أو EMPTY إذا لم تنته اللعبة. إذا فازت اللعبة ، نرمي برسالة تشير إلى السبب. بهذه الطريقة ، IllegalStateException نمنع اللاعب من وضع علامة بعد أن تم تحديد اللعبة. |

5	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?
		
		الاختلاف الرئيسي بين اختبار المساواة الضحلة واختبار المساواة العميقة يكمن في كيفية مقارنة العناصر داخل المصفوفات.
		
		• اختبار المساواة الضحلة: يفحص فقط ما إذا كان كلا المصفوفين يشيران إلى نفس كائن المصفوفة في الذاكرة. وهذا يعني أنه يتحقق ما للمصفوفين.
		• اختبار المساواة العميقة: يفحص قيم العناصر داخل المصفوفين. وهذا يعني أنه يتحقق ما إذا كان كل عنصر في نفس الفهرس في كلا المصفوفين متساويًا.
		
		بالنسبة للمصفوفات ذات البعد الواحد، يكون الاختبار بسيطًا، ولكن بالنسبة للمصفوفات ذات الأبعاد المتعددة، يصبح الاختبار أكثر تعقيدًا ويتطلب مقارنة قيم العناصر داخل كل مصفوفة فرعية.
		
		بشكل عام، تكون اختبارات المساواة العميقة أكثر دقة وضرورية لتحديد المساواة الحقيقية بين المصفوفات المحتوية على عناصر ذات قيم مختلفة أو هياكل متداخلة.
		
		`public class Q5 {`
		`int[] arr1 = {1, 2, 3};`
		`int[] arr2 = {1, 2, 3};`
		
		`boolean shallowEqual = arr1 == arr2; // true`
		`boolean deepEqual = arr1.length == arr2.length;`
		
		`for (int i = 0; i < arr1.length; i++) {`
		`if (arr1[i] != arr2[i]) {`
		`deepEqual = false;`
		`break;`
		`}`
		`}`
		
		`System.out.println("Deep equal: " + deepEqual); // true`
		`int[][] arr1 = {{1, 2, 3}, {4, 5, 6}};`
		`int[][] arr2 = {{1, 2, 3}, {4, 5, 6}};`
		
		`boolean shallowEqual = arr1 == arr2; // true`
		`boolean deepEqual = arr1.length == arr2.length;`
		
		`for (int i = 0; i < arr1.length; i++) {`

```
        if (arr1[i].length != arr2[i].length) {
            deepEqual = false;
            break;
        }

        for (int j = 0; j < arr1[i].length; j++) {
            if (arr1[i][j] != arr2[i][j]) {
                deepEqual = false;
                break;
            }
        }
    }

System.out.println("Deep equal: " + deepEqual); // true
}
```

6	R-3.14	Give 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.

هناك ثلاث طرق مختلفة لإنشاء نسخة من مصفوفة في Java:

1. استخدام `System.arraycopy`: تقوم هذه الطريقة بنسخ العناصر من مصفوفة إلى أخرى باستخدام طريقة `System.arraycopy`. تأخذ هذه الطريقة خمسة وسيطات:
   - `original`: مصفوفة المصدر.
   - `0`: الفهرس الأول للنسخ من مصفوفة المصدر.
   - `backup`: مصفوفة الهدف.
   - `0`: الفهرس الأول للنسخ في مصفوفة الهدف.
   - `original.length`: عدد العناصر المراد نسخها.
2. استخدام `Arrays.copyOf`: تقوم هذه الطريقة بإنشاء مصفوفة جديدة بنفس طول مصفوفة المصدر ونسخ جميع العناصر منها إلى المصفوفة الجديدة. تأخذ هذه الطريقة وسيطتين:
   - `original`: مصفوفة المصدر.

		o `original.length`: طول المصفوفة الجديدة.
		
		3. استخدام حلقة `for`: تقوم هذه الطريقة بنسخ كل عنصر من مصفوفة المصدر إلى المصفوفة الجديدة بشكل فردي. تقوم بإنشاء مصفوفة جديدة بنفس طول مصفوفة المصدر ثم تكرر كل عنصر في مصفوفة المصدر وتقوم بتعيين قيمته إلى العنصر المقابل في المصفوفة الجديدة.
		
		إن جميع الطرق الثلاثة تعمل على إنشاء نسخة كاملة من المصفوفة الأصلية، ولكنها تختلف في طريقة التنفيذ. الطريقة الأولى هي الأكثر كفاءة من حيث الذاكرة، بينما الطريقة الثانية هي الأكثر سهولة في الاستخدام. والطريقة الثالثة هي الأكثر مرونة إذا كنت بحاجة إلى إجراء أي معالجة على العناصر أثناء نسخها
		
		`public class Q14 {`
		`int[] original = {1, 2, 3, 4, 5};`
		`int[] backup = new int[original.length];`
		`System.arraycopy(original, 0, backup, 0, original.length);`
		`int[] original = {1, 2, 3, 4, 5};`
		`int[] backup = Arrays.copyOf(original, original.length);`
		`int[] original = {1, 2, 3, 4, 5};`
		`int[] backup = new int[original.length];`
		`for (int i = 0; i < original.length; i++) {`
		`backup[i] = original[i];`
		`}`
		`}`

7	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. خوارزمية لإيجاد المكرر في مصفوفة (أ) بحجم n> = 2 إلى ١ مع أعداد صحيحة من n-1: 1. احسب مجموع جميع العناصر في المصفوفة (مجموع_مصفوفة): مجموع العناصر. 2. احسب المجموع المتوقع للمصفوفة إذا لم يكن بها أي تكرارات: المجموع المتوقع (مجموع_متوقع = n * (n-1) / 2). 3. احسب الفرق بين مجموع العناصر والمجموع المتوقع (الفرق = الفرق: مجموع_مصفوفة - مجموع_متوقع). 4. العنصر المكرر هو الفرق: العنصر المكرر.

		التفسير: مرة واحدة n-1 نظرًا لأن المصفوفة تحتوي على جميع الأعداد الصحيحة من ١ إلى n * (n-1) / 2. بالضبط، يجب أن يكون مجموع جميع العناصر إذا تم تكرار أي عنصر، فسوف يساهم بقيمته مرتين في المجموع، مما يؤدي إلى اختلاف بين المجموع الفعلي والمتوقع. سيكون هذا الاختلاف هو قيمة العنصر المكرر. مثال: دع A = [1, 2, 3, 4, 2, 7, 8, 8, 3]. 1. مجموع العناصر: مجموع_مصفوفة = ١ + ٢ + ٣ + ٤ + ٢ + ٧ + ٨ + ٨ + ٣ = ٣٨. 2. المجموع المتوقع: مجموع_متوقع = ٩ * ٨ / ٢ = ٣٦. 3. الفرق: الفرق = مجموع_مصفوفة - مجموع_متوقع = ٣٨ - ٣٦ = ٢. 4. العنصر المكرر: العنصر المكرر هو ٢. التعقيد: • تعقيد الوقت: O (n) بسبب الحلقة الفردية لحساب المجموع. • تعقيد المساحة: O (1) حيث يتطلب مساحة إضافية ثابتة.
8	C-3.18	Let B be an array of size $n \geq 6$ 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. خوارزمية لإيجاد خمسة أعداد مكررة في مصفوفة B: 1. خريطة التردد: قم بإنشاء قاموس فارغ (frequency_map) لتخزين تردد كل عدد صحيح يتم مواجهته. 2. التكرار: (ب) تكرر خلال كل عنصر في المصفوفة.

3. فحص التردد: لكل عنصر (x) في B:
   - إذا كان x موجودًا في frequency_map:
     - قم بزيادة تردد x في القاموس.
   - خلاف ذلك:
     - أضف x إلى القاموس بتردد ١.
4. ابحث عن العناصر في frequency_map بتردد أكبر من ١: تحديد التكرار هذه هي الأعداد الخمسة المكررة.

التفسير:

تكرر هذه الخوارزمية خلال المصفوفة وتتبع تردد كل عدد صحيح يتم مواجهته في قاموس. إذا ظهر عدد صحيح أكثر من مرة، سيتم زيادة تردده. أخيرًا، نحدد الأعداد الصحيحة بتردد أكبر من ١، والتي تمثل الأعداد الخمسة المكررة.

التعقيد:

- تعقيد الوقت: O (n) بسبب الحلقة الفردية للتكرار عبر المصفوفة.
- تعقيد المساحة: O (n) بسبب استخدام خريطة التردد.

مثال:

دع B = [1, 2, 3, 4, 5, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 3, 4, 5].

1. قم بتعريف قاموس فارغ (frequency_map).
2. تكرر خلال كل عنصر في B:
   - 1: إذا كان ١ موجودًا في frequency_map ، قم بزيادة تردده. وإلا ، أضفه بتردد ١.
   - 2: إذا كان ٢

9	C-3.19	Give Java code for performing add(e) and remove(i) methods for the Scoreboard class, as in Code Fragments 3.3 and 3.4, except this time, don't maintain the game entries in order. Assume that we still need to keep n entries stored in indices 0 to n−1. You should be able to implement the methods without using any loops, so that the number of steps they perform does not depend on n. 1. تحقق مما إذا كانت هناك مساحة لإدخال جديد (numEntries < board.length). 2. إذا كانت الإجابة بنعم، أضف الإدخال الجديد في الفهرس المتاح التالي (numEntries) وقم بزيادة numEntries. إزالة (i): 1. تحقق ما إذا كان الفهرس صالحًا (i >= 0 و i < numEntries).

2. قم بتخزين الإدخال في الفهرس المحدد في متغير مؤقت.
3. قم باستبدال الإدخال في الفهرس المحدد بالإدخال في الموضع الأخير (numEntries - 1).
4. قم بتعيين الإدخال الأخير إلى null.
5. قم بتقليل numEntries.
6. قم بإرجاع الإدخال الذي تمت إزالته.

هذه الطريقة أكثر كفاءة من استخدام الحلقات لأنها لا تتطلب أي عمليات تكرر إضافية، مما يجعلها تعمل بشكل أسرع مع أي عدد من الإدخالات.

```
public class Scoreboard {

        private int numEntries; // number of actual
entries
        private GameEntry[] board; // array of game
entries (names & scores)

        public Scoreboard(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;
        }

}
```

10	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.
11	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? 1. نطاق القيم: يتم إنشاؤه باستخدام r.nextInt(10)، نظرًا لأن كل عنصر في A يمكن أن يكون كل عنصر أي عدد صحيح من ١ إلى ١٠. 2. ضرب العناصر: حاصل ضرب جميع العناصر في A، يمكن أن تتراوح قيمة x من ١ (إذا كانت جميع العناصر ١) إلى ١٠^١٠٠ (إذا كانت جميع العناصر ١٠). 3. القيمة الأكثر احتمالاً: ستكون القيمة الأكثر احتمالية لـ x هي الرقم الذي يظهر بشكل متكرر كحاصل ضرب مجموعة فرعية من العناصر في A. الحل: 1. ملاحظة: نظرًا لوجود طرق أكثر بكثير للحصول على حاصل ضرب ١ من أي حاصل ضرب آخر، فإن ١ سيكون القيمة الأكثر احتمالية لـ x. 2. الحساب: لحساب احتمال أن تكون x = 1: o عدد طرق الحصول على حاصل ضرب ١: ١٠٠^١ (جميع العناصر هي ١). o العدد الإجمالي لطرق إنشاء A: 10^100 (كل عنصر لديه ١٠ احتمالات). o احتمال أن تكون x = 1: 1^100 / 10^100 = 1/10^100. لذلك، فإن الرقم الأكثر احتمالاً ليكون قيمة x هو ١/١٠^١٠٠، واحتمال حدوث ذلك هو ١.
12	C-3.22	Write a method, shuffle(A), that rearranges the elements of array A so that every possible ordering is equally likely. You may rely on the nextInt(n) method of the java.util.Random

class, which returns a random number between 0 and n−1 inclusive.

المعروفة لخلط العناصر. إليك طريقة Fisher-Yates هذه الطريقة تستخدم خوارزمية عملها:

1. بدءًا من العنصر الأول (A) تكرر خلال كل عنصر من المصفوفة: التكرار.
2. ضمن (j) ، اختر فهرسًا عشوائيًا(i) بالنسبة للعنصر الحالي: اختيار فهرس عشوائي هذا يضمن أن كل (إلى النهاية i من) الجزء المتبقي غير المخلط من المصفوفة عنصر لديه فرصة متساوية للوقوع في أي موضع.
3. بالعنصر الموجود في (i) قم بمبادلة العنصر في الفهرس الحالي: تبديل العناصر الفهرس العشوائي (j).
4. استمر في التكرار وتبديل العناصر حتى تتم معالجة جميع العناصر: التكرار.

```
public class Q6 {
    import java.util.Random;

    public class Shuffler {

        public static void shuffle(int[] A) {
            Random random = new Random();

            for (int i = 0; i < A.length; i++) {
                // Choose a random index j between i and
A.length-1 (inclusive)
                int j = random.nextInt(A.length - i) + i;

                // Swap the elements at indices i and j
                swap(A, i, j);
            }
        }

        private static void swap(int[] A, int i, int j) {
            int temp = A[i];
            A[i] = A[j];
            A[j] = temp;
        }

        // Example usage
        public static void main(String[] args) {
            int[] A = {1, 2, 3, 4, 5};
            shuffle(A);

            for (int i : A) {
                System.out.print(i + " ");
            }
        }
    }

}
```

13	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. طريقة لتتبع أزواج اللاعبين المتقابلين والفائز في لعبة الغابة متعددة اللاعبين: هياكل البيانات: 1. مصفوفة اللقاء: مصفوفة منطقية ثنائية الأبعاد met بحجم n × n ، حيث met[i][j] صحيح إذا كان اللاعبان i و j قد التقيا، وإلا فهو خاطئ. 2. مصفوفة عدد اللقاءات: مصفوفة meeting_count بحجم n ، حيث meeting_count[i] تخزن عدد اللاعبين الذين التقاهم اللاعب i. 3. متغير الفائز: متغير winner لتخزين معرف اللاعب الفائز، مع تهيئته إلى None. الوظائف: 1. meet(i, j): يتم استدعاء هذه الطريقة كلما التقى اللاعبان i و j لأول مرة. تقوم بتحديث مصفوفة met ومصفوفة meeting_count: • يتم تعيين met[i][j] و met[j][i] إلى true. • يتم زيادة meeting_count[i] و meeting_count[j]. 2. check_winner(): يتم استدعاء هذه الطريقة بشكل دوري أو بعد كل اجتماع للتحقق مما إذا كان أحد اللاعبين قد التقى بالجميع: • كرر خلال كل لاعب i. • إذا كان meeting_count[i] يساوي n-1، (التقى بجميع اللاعبين الآخرين) فقم بتعيين winner إلى i وقم بالتوقف.

		3. get_winner(): تُرجع هذه الطريقة معرف الفائز، أو None إذا لم يتم العثور على فائز حتى الآن. التفسير: تخزن مصفوفة met معلومات بكفاءة حول جميع لقاءات اللاعبين. تساعد مصفوفة meeting_count في التعرف بسرعة على اللاعبين القريبين من الفوز. تحقق طريقة check_winner من وجود فائز بعد كل اجتماع أو بشكل دوري. أخيرًا، توفر طريقة get_winner طريقة سهلة للوصول إلى معرف الفائز.

14	C-3.24	Write a Java method that takes two three-dimensional integer arrays and adds them componentwise.
		
		1. التحقق من الإدخال: تتحقق الطريقة أولاً مما إذا كان للمصفوفتين نفس الأبعاد. إذا لم يكن الأمر كذلك، فإنه يلقي استثناء IllegalArgumentException. 2. مصفوفة النتيجة: ثم تقوم بإنشاء مصفوفة ثلاثية الأبعاد جديدة result بنفس أبعاد مصفوفات الإدخال. 3. تكرار الحلقة: تكرر الطريقة خلال كل عنصر من مصفوفي الإدخال باستخدام حلقات متداخلة المضمنة. 4. الإضافة المكونة للمكونات: لكل عنصر، تضيف الطريقة العناصر المقابلة من مصفوفي الإدخال وتخزن المجموع في العنصر المقابل لمصفوفة result. 5. إرجاع النتيجة: أخيرًا، تُرجع الطريقة المصفوفة result التي تحتوي على مجموع مصفوفي الإدخال.
		
		`public class Q7 {`
		`    public static int[][][] add3DArrays(int[][][] array1, int[][][] array2) {`
		`        if (array1.length != array2.length || array1[0].length != array2[0].length || array1[0][0].length != array2[0][0].length) {`
		`            throw new IllegalArgumentException("Arrays must have the same dimensions.");`
		`        }`
		
		`        int[][][] result = new int[array1.length][array1[0].length][array1[0][0].length];`

```
        for (int i = 0; i < array1.length; i++) {
            for (int j = 0; j < array1[i].length; j++) {
                for (int k = 0; k < array1[i][j].length;
k++) {
                    result[i][j][k] = array1[i][j][k] +
array2[i][j][k];
                }
            }
        }

        return result;
    }
}
```